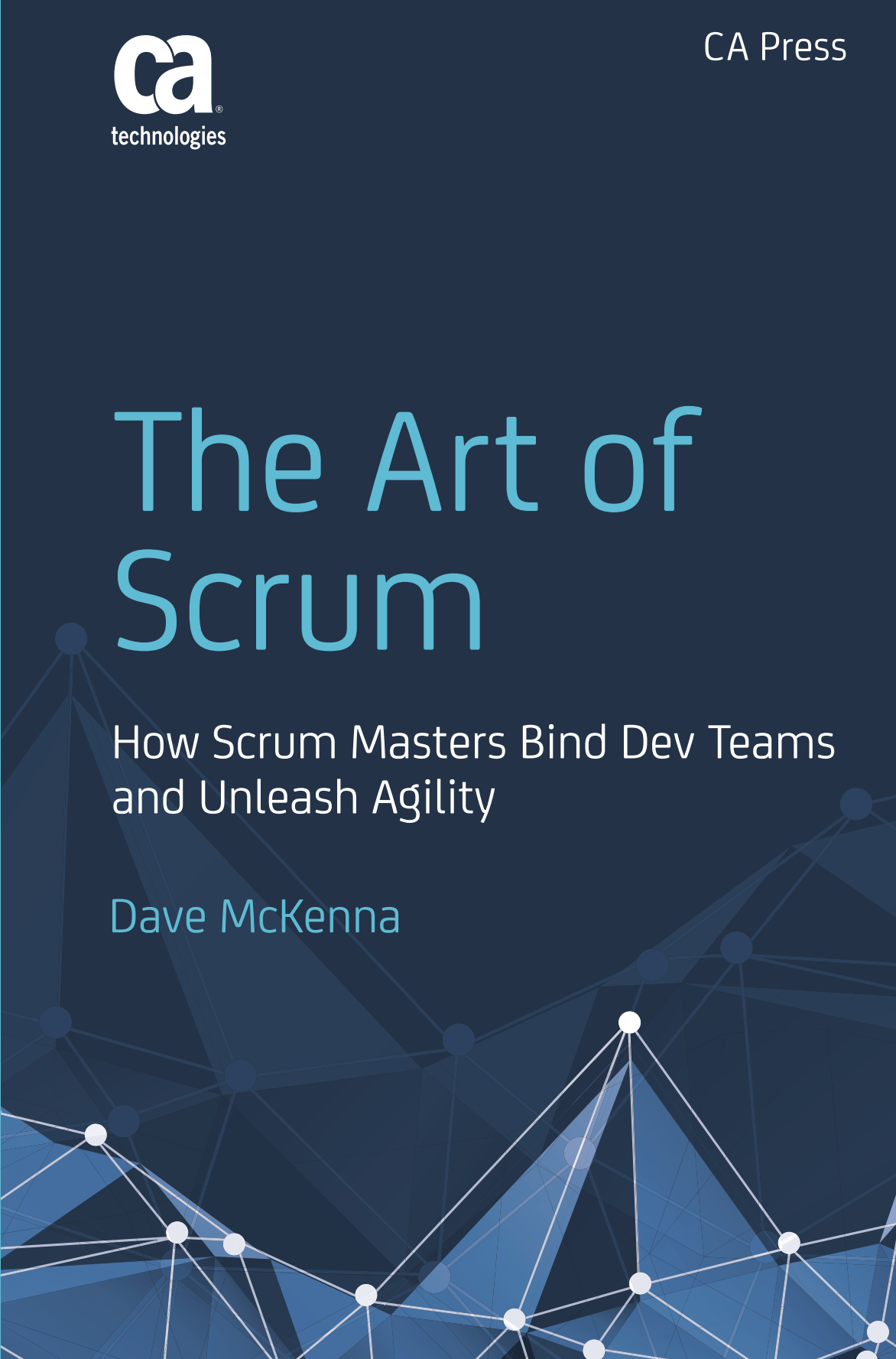
ca
technologies
CA Press
The Art of Scrum
How Scrum Masters Bind Dev Teams and Unleash Agility
Dave McKenna

THE ART OF SCRUM

HOW SCRUM MASTERS BIND DEV TEAMS AND UNLEASH AGILITY

Dave McKenna

The Art of Scrum: How Scrum Masters Bind Dev Teams and Unleash Agility

Dave McKenna
CA Technologies, Aliquippa, Pennsylvania,
USA

ISBN-13 (pbk): 978-1-4842-2276-8 ISBN-13 (electronic): 978-1-4842-2277-5
DOI 10.1007/978-1-4842-2277-5

Library of Congress Control Number: 2016958011

Managing Director: Welmoed Spahr
Acquisitions Editor: Robert Hutchinson
Developmental Editor: James Markham
Technical Reviewer: Elaine Ritchie
Editorial Board: Steve Anglin, Pramila Balen, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Rita Fernando
Copy Editor: James A. Compton, Compton Editorial Services
Compositor: SPi Global
Indexer: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Printed on acid-free paper

To my Father, who passed from this life to the next while I was writing this book. Dad, I hope that somehow you know that I finished this book.

Contents

About the Author vii
About the Technical Reviewer ix
Acknowledgments xi
Introduction xiii

Part I: Scrum: Overview 1
Chapter 1: The Agile Principles 3
Chapter 2: The Scrum Framework 27
Chapter 3: Scrum Roles 35
Chapter 4: Scrum Team Structures 55
Chapter 5: Scrum Ceremonies and Artifacts 63

Part II: Scrum: The Scrum Master's Perspective 95
Chapter 6: The Scrum Master's Responsibilities and Core Functions . . . 97
Chapter 7: The Scrum Master's Interaction with Other Roles 133

Part III: The Scrum Master's Skill Sets 147
Chapter 8: Soft Skills of the Scrum Master 149
Chapter 9: Technical Skills of the Scrum Master 163
Chapter 10: Contingency Skills of the Scrum Master 181
Chapter 11: Putting It All Together 189

Index 211

About the Author

Dave McKenna is a certified Scrum Master and Agile Coach who has worked in the information technology field for more than twenty years. A United States Air Force veteran who started his career in IT unboxing IBM and Apple computers in a Computerland store, McKenna eventually became a Novell Certified Network Engineer and worked his way into a Sustaining Engineer position at CA Technologies. In 2009 he began his Scrum Master journey, which continues today.

About the Technical Reviewer

Elaine Ritchie is an Agile Coach and SAFe Program Consultant at CA Technologies with over 30 years of experience in software development. With a degree in Computer Science and a background in software engineering, management, product ownership, and project management, Elaine held almost every role in Scrum before becoming a Scrum Master and Agile Coach. Since 2013 she has been leading CA's Mainframe division through Agile Transformation through coaching, teaching, and facilitation.

Elaine is a Certified Scrum Master (CSM), an ICAgile Certified Professional in Agile Coaching (ICP-ACC), and a SAFe Program Consultant (SPC4).

Acknowledgments

I'd like to thank my family for their unwavering support through the writing of this book. When it got tough (and it did), it was you guys that kept me going.

Thank you to my fellow Agile coaches Elaine Ritchie and Jay Cohen. When you read this book, it's going to sound more like Elaine and Jay talking than me. That's because they constantly have poured into me during my Agile journey. Without them, this book would not have happened.

Thank you to Bob Carpenter for believing in me. It was Bob who encouraged me to become a Scrum Master and later a coach. I owe my career to him.

Thank you to Jake Turner and Tony Pharr. Jake showed me what an impact a coach could make on somebody's life, and Tony showed me how a champion goes about his business.

Thank you to all the teams I've worked with in my career. This book is for you all. All of the high points and low points of our combined experiences have woven a beautiful tapestry that I have tried to describe in these pages.

Thank you to the editors Rita Fernando, Robert Hutchinson, and James Markham. I'm sure my unique writing style presented some challenges. I appreciate all of your hard work.

Last, I'd like to thank my best friend and wife Rhonda for putting up with me. Believe me, I don't know how she does it…

Introduction

It's December 2006 and I'm sitting on a 5-gallon bucket in front of a Best Buy store. It's 2 am and it's cold—really, really cold. I have my father's hunting jacket on, so I look like a 300-pound pile of leaves while I'm sitting there. Snow starts to fall softly as I look at the other folks who've lined up outside this electronics retailer, and I can't help but keep asking myself the same thing over and over again.

"Why am I doing this?"

It's dark, it's cold, and I look like I belong on an episode of Duck Dynasty more than six years before the first episode aired. Why am I spending the night in front of a multichannel consumer electronics retailer?

Three words: A Nintendo Wii.

It seemed like that Christmas the entire Western Hemisphere wanted one of those game systems—including my daughters. If there was going to be a big surprise on the morning of December 25th, I was going to have to hang out on the sidewalk and hope what I saw on the internet was accurate. This was before the days of social media. I spent a good deal of my time intently monitoring forums and chatrooms. On this day, the word was that this location would have 15 of the technological marvels… I was willing to do whatever it took to get one.

I had tried to order it from a number of online retailers, but each one gave the same response. The Wii would not be shipped until after December 25th. So there I was, sitting on a bucket, ninth in line. When I showed up around 10 pm there were already eight people there, and by the time the store manager came out at 1 am, fifteen folks (including yours truly) were lined up. He didn't tell us how many Wii's he would have in the morning. He just said that he did have them, and they would pass tickets out at 7 am. You would think that tension would be high for the next six hours. That we would be eyeing each other like Old West gunslingers. The slightest move in the wrong direction and all hell would have broken loose. After all, these things were impossible to get your hands on. This may have been the last chance to get one before that fat, jolly elf came down the chimney.

But that wasn't the case. We all sat around and talked about the cool things we heard the Wii could do. Folks were so amped up that they were positively giddy about the fact that they were going to get one. Folks were blown away

by the fact that you would not be mashing buttons with this new console. It sensed your movement, and reacted accordingly. It was so innovative that it was selling rings around its holiday competition, the Sony Playstation 3. In fact, when I was out hunting a Wii, there was always a stack of PS3s sitting there. Every time I asked a sales associate if they had any Wii consoles, the answer was always, "No, but I have Playstations if you want one."

This, ladies and gentlemen, is the reason I decided to become a Scrum Master. This behavior is what we should be looking for from our customers. No, I don't want customers camped out on the sidewalk in front of my office (although I do admit that it would be really cool if it ever did happen…).

What I do want to see is customers who cannot wait for the next release of our products. Customers who attend our Sprint review meetings and say "I want that as soon as possible… Can I have it Now?!".

Customers who are so intimately involved with the Agile development process that they can't help but get excited about what the Scrum teams are working on. Because they have communicated to product management what they need to get from the release, and the Scrum team delivered that functionality while getting constant feedback from them.

A small man, who looks a bit like Danny DeVito, is trying to convince a skeptical room full of developers and management types that this thing called "Agile" will work at our company. I really did not see how this was going to work. He was talking about increasing customer interaction, which I was all for. Working as a sustaining engineer, I was frustrated with what we were putting in our products. In fact, I had just sat in a meeting where a customer took our development manager to task for not getting their enhancements into the product. The customer wasn't upset because it was taking too long to get what they asked for—they were upset because none of the stuff they had asked for was planned to be in any future release. They felt ignored and were threatening to go to a competitor's product.

The one thing that really hit home about this Agile stuff that was being presented was the fact that the customer was really the focus… of everything. Development would be done on "Scrum teams," which were self-organized, cross-functional teams that are hyper-focused on delivering customer value. In order to do this, the command-and-control structure that was in place would have to change. New Agile roles would need to be created. One that grabbed my interest was a role called "Scrum Master."

As it was explained to us way back then by that wild little guy, the job of Scrum Master could have a big impact on costs, saving the company money —period. The role was a full-time position. A Scrum Master is the servant-leader of the Scrum team. At the risk of being overly simplistic, the Scrum Master is responsible for the Agile process, doing whatever it takes to ensure that the team embraces Agile and is constantly improving.

Yes, it is a full-time role. Many folks really have a hard time wrapping their minds around that one. I mean, what does a Scrum Master do all day to justify a full-time role?

Removing impediments and dealing with problems is always at the top of the list, because impediments or problems are always there—especially for teams in large companies. What are impediments? It could be that another team does not respond in a timely fashion, or your team needs to understand how an interface works and the people who wrote it are not available (or even working for the company any more), or you have someone out sick for a few days.

You also need to interface with management, doing everything from convincing them that an Agile process is worthwhile to reminding them that the team is responsible for delivering value, not lines of code. Did I mention that in order for Agile to work, the old command-and-control nature of management needs to be eliminated? That leads to some interesting, albeit courageous conversations with people who have a direct influence on your paycheck.

Finally, you work to help the team solve its own problems and remove its own impediments. A good Scrum Master should be looking to work himself out of a job by creating a high-performing team. For some teams, this may be impossible to achieve. However, it should always be the goal. A mature team is truly a self-managing entity.

A Scrum Master needs to be more than a bulldozer. She needs to truly care for each and every member of the team. People are human. Sometimes we are dealing with stuff at home, or walking into the coffee room and are greeted by an empty pot, or have a terrible drive into the office, or are just in a bad mood. Sometimes folks just lose their minds for no reason. Dealing with this situation takes coaching and people skills. A Scrum Master needs to know when to jump in and when to back off. Sometimes getting somebody coffee when they are busy or remembering their birthday is all it takes to make them feel like part of the team. It is the Scrum Master's responsibility to ensure that everybody on his or her team is happy.

A Scrum Master must be able to introduce new ideas and get the team to try new things. Change can be very frightening, but it also can be rewarding.

I am probably the last person you would expect to be leading a Scrum team. I'm a 300-pound, bearded, tattooed, mountain of a man who looks more like a bouncer than a Scrum Master. In fact, I have been a bouncer in my quite colorful past. I think I have turned into a pretty good Scrum Master. This book will explain the Scrum framework, the Scrum roles, and how a Scrum Master can impact everybody and help deliver value to customers and stakeholders faster than anybody ever thought possible.

In the first part of this book, I talk about the old Waterfall development model, the Agile Manifesto, and the 12 principles. I go over Scrum and the Scrum roles. In the middle of the book I focus more on the Scrum Master. I even introduce you to a hypothetical scrum master and team to illustrate what it looks like when a team starts to embrace Scrum. The last part if the book is where I delve deeper into the Scrum Master role.

I hope you will find this a valuable resource. My writing style is a bit all over the place. At the very least, I hope you find it entertaining.

PART

I

Scrum: Overview

CHAPTER

1

The Agile Principles

Welcome to the fourth industrial revolution

When I was in grade school, I read books about what could be expected to happen in the twenty-first century. It was predicted that things like flying cars, one-piece suits that looked like they were made of aluminum foil, and vacations on Mars would be commonplace by now. None of that stuff has come to pass yet, but there are some amazing things happening. Nobody is happier than me that cars can now parallel park by themselves. Cell phones now pack more computing power than early mainframes. In fact, the primary function of a smartphone is no longer making or receiving phone calls. People are using 3D printers to print robotic, prosthetic limbs.

And software runs it all.

We live in an age where software literally touches everything. It is the great equalizer. Software is not built just by traditional software companies, but by companies like banks, insurance companies, car manufacturers—you name it and software is an integral part of it.

Welcome to the fourth Industrial Revolution…

Time for a little history lesson. The Agricultural Revolution occurred when human beings started planting crops and domesticating animals. They became less nomadic. In other words, people stopped wandering around and stayed in one place. They worked the land and herded animals.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_1

The Industrial Revolution caused people to leave their farms. Humans started working with machines in factories. Villages were abandoned for big cities. Factory work was structured, measured, and steeped in command and control.

The Information Revolution relies on knowledge workers. The term *knowledge worker* does not just refer to software developers. Anybody who handles or uses information can be classified as a knowledge worker—— folks like doctors, teachers, and scientists. Knowledge work is different. For example, the emphasis is more on changing things than running things. Less structure and continuous innovation is required as the work is constantly changing and evolving.

The fourth Industrial Revolution refers to quickly emerging, disruptive technologies that fundamentally will not only change the way we live and work, but how we relate to each other. The ability to collaborate, pivot, and adapt to change is critical to survival in the age in which we now find ourselves. Technology is everywhere and is impacting both business and society. The lines between physical and digital are being blurred—requiring, dare I say, agility to survive and thrive.

What Animal do You Think of When I Say the Word "Agile"?

When I ask that question, most folks pick the cheetah. On the surface, that is a great choice; however, I think that what the furry speed merchant is trying to chase down may actually be the best example of agility in the animal kingdom.

The cheetah is the fastest animal on land. No doubt about that. The cheetah's favorite meal, the Impala antelope, is almost as fast but can run at top speed for a longer time. A cheetah can only run all-out for around twelve seconds or so. To keep from becoming dinner, the antelope needs to hold the cat off until it cannot sustain the effort required to keep up the frantic pace. The impala will jump and change direction quickly and force the cheetah not to pursue in a straight line, where its superior speed would enable a quick kill. The antelope tries to tire out the sleek cat by using superior agility. In the animal kingdom, agility is vital for survival.

Agility is About Moving and Adjusting Quickly

In my freshman year, I believe it was 1981, I had fractured my ankle on the first day of practice in pads during the first play I tried to run as a quarterback. As I learned, the weakness of a four-four defense was either up the middle or around the end. I took the ball around the right of the defense for a 30-yard gain. I focused on the air whistling through my helmet. Running like the wind… until I stepped in a gopher hole. As the defense caught up to me, I hear the head coach yell, "If that was a good quarterback he would have scored!"

It took a long time for my ankle to heal, so I did not play during my sophomore year, On top of that, I wasn't exactly what you would call graceful. I would say that I was gangly, not used to my six-foot frame. My Uncle Frank, who played for the University of Nebraska and was trying to help me with my ill-fated high school football career, had a different term he used to describe my movement patterns:

Heavy Footed.

"What do you see boxers doing all of the time?" My uncle asked.

"Jumping rope," I replied.

"That's right!" he said. "Every chance they get, those guys jump rope. That is because footwork is important to a boxer. Same thing with football. You need to do a lot of agility drills like jumping rope because you can't move your feet."

I jumped rope, lifted weights, and ran, but the football thing never panned out for me. I find it somewhat ironic that the person who could not get out of his own way now coaches Agility.

No, not that kind of agility…

If you challenged me to describe the Agile framework in one sentence, I would say this:

Agile is about adapting to change, not planning everything out up front.

Agile is many things. Iterating often, freeing development up to do what they do best, aligning the business with software, trying to produce stuff that our customers really want to buy, and developing a culture of continuous improvement——All great stuff that is part of Agile. However, to me it is all about responding to change.

In my early days on a development team, change was a disruptive force that was about as welcome as a rabid raccoon in your garage. Since all the planning was done at the beginning of a project, any type of change created problems. A team needed to avoid change at all cost.

However, in reality the change we were avoiding was a customer that needed help. It was a flaw with a design that needed addressed, it was harnessing a new technology to make a product better. The requirements change so quickly that it is foolish to try to plan out a large-scale release up front.

Teams need to embrace change. They need to shift the way they work. Change is going to happen. Agile teams need to crave change. Sticking to the plan is not going to cut it anymore. The power is in the ability to pivot and adapt quickly. That is how we create delighted customers.

Who Doesn't Like Waterfalls?

When I started working for a software vendor, I did not know that we were using the Waterfall methodology; I just thought it was the way we did things. The Waterfall methodology has goals for each phase of development. Once a phase of development is completed, you move on to the next phase. You cannot turn back once you have passed through a phase. Since we are talking about waterfalls, let us look at probably the most famous waterfall in the world, Niagara Falls. When you go there, the beauty and majesty of the place makes an impact. The sheer volume of water that passes over the falls every day is amazing. One thing that is obvious is that once water goes over the cliff you cannot get it back up the mountain. The same with Waterfall development. As you see in Figure 1-1, you cannot turn back.

Figure 1-1. Once water goes over the falls, there is no going back

When you think about how you build software, the Waterfall model describes a method that is linear, gated, and sequential. You take a development project and you break it up into stages or phases of development. Each one of these phases has a goal. You move onto the next phase once you achieve the goal. I have seen Waterfall described as resembling a track and field relay race. A runner takes a baton, runs his leg of the race, and passes the baton to the next runner.

Figure 1-2 shows the phases of the Waterfall method.

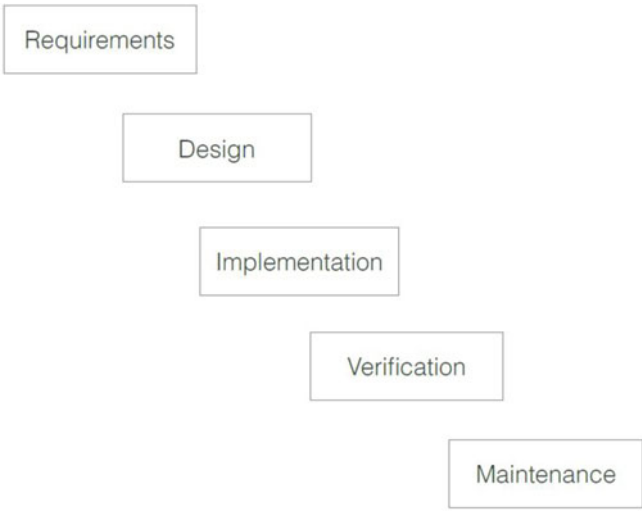

Figure 1-2. Waterfall phases

The Requirements Phase

A team will start with the requirements phase, as shown in Figure 1-3. This is where business analysts or product marketing define the requirements.

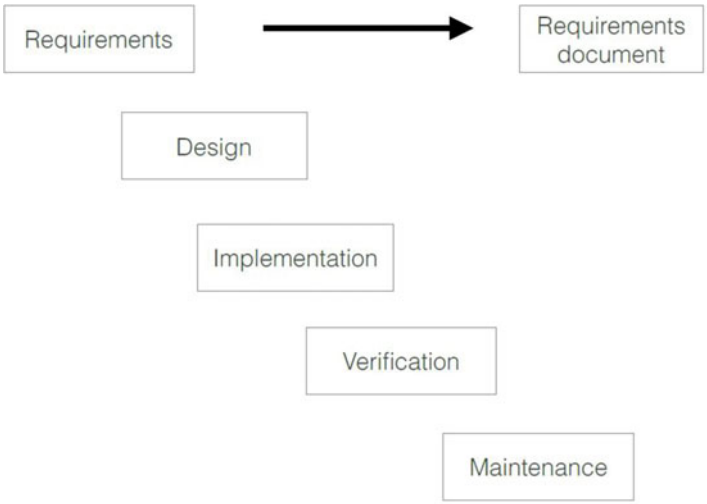

Figure 1-3. The Waterfall Requirements phase

In other words, they would figure out what we were going to build, and what both customers and the business require out of this project. The goal for the requirements phase is usually some kind of requirements document that details the team's findings.

As shown in Figure 1-4, the team did all planning in the design phase. Design work consists of stuff like deciding how the team will write the code. They would figure out the languages and techniques to use, to best deliver functionality and value, and even who would work on what. The result of all this work would be the development staff producing an internal requirements document, as shown in Figure 1-5.

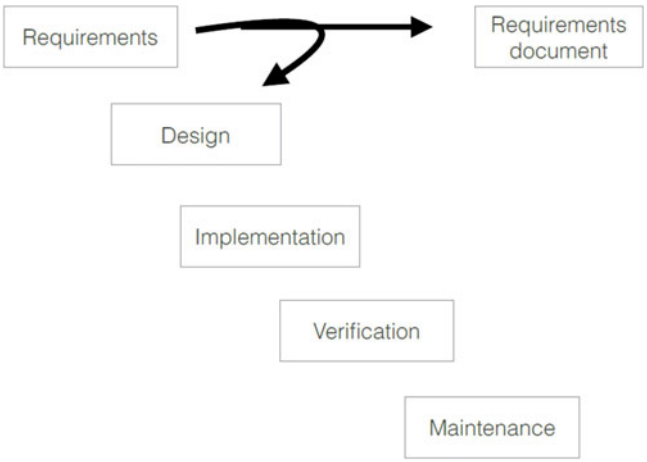

Figure 1-4. The Waterfall Design phase

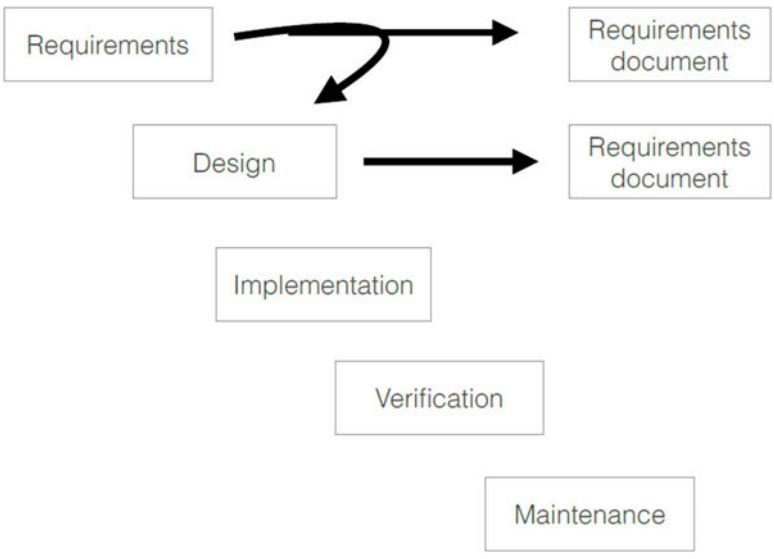

Figure 1-5. The Waterfall requirements document

The Implementation Phase

The Implementation phase, as shown in Figure 1-6, is where developers do what they do best. They write the code and transform ideas into functional product. Developers take the requirements as defined in the Requirements phase and create value for customers. At the end of the implementation phase, software is produced, and that software resembles a viable product.

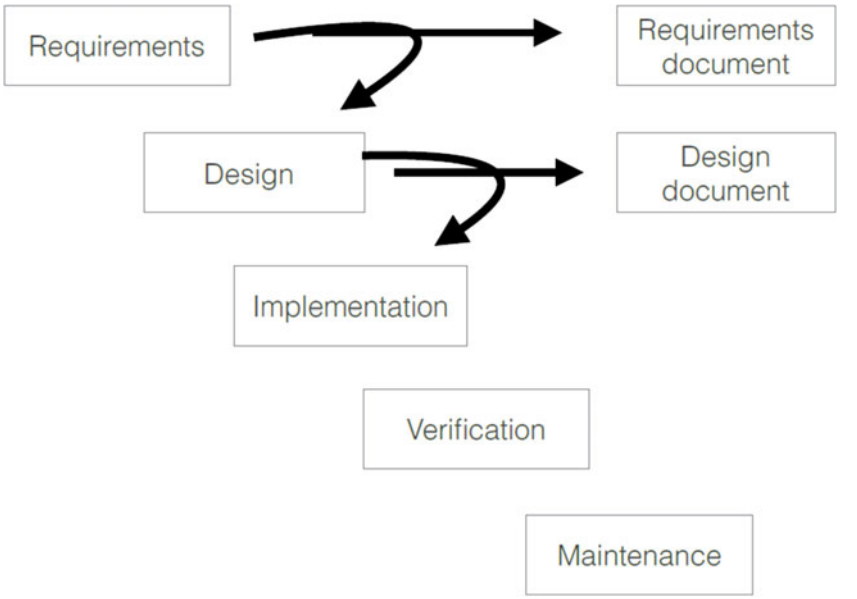

Figure 1-6. The Waterfall Implementation phase

The Verification Phase

We have software at this point, and that is a good thing. However, software needs to be in good working order. Figure 1-7 shows the Verification phase, where the Quality Assurance engineers test the software. They work to ensure that the team delivers a high-quality product to the customers. The QA engineers take the design document produced during the design phase and create test plans from it. They execute the test plans against the software produced in the Implementation phase. The development staff addresses any defects found. The quality assurance engineers work as a separate team. They are not considered part of development.

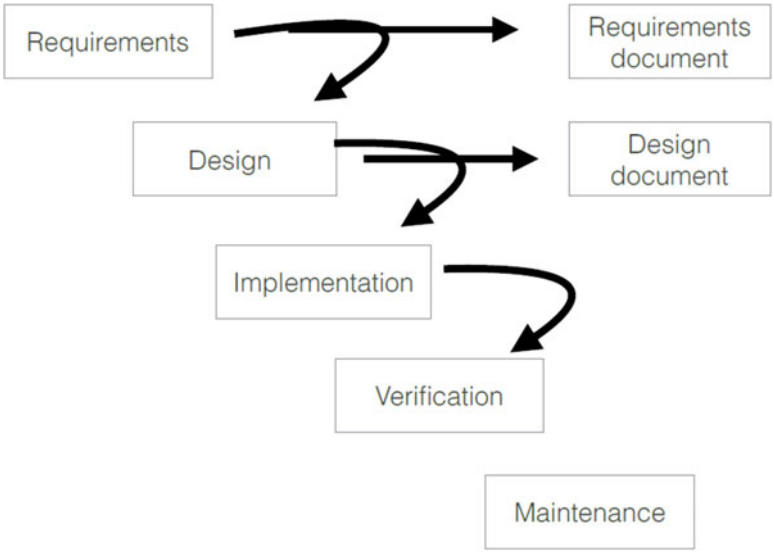

Figure 1-7. The Waterfall Verification phase

Once the software is thoroughly tested and everybody agrees that it is in a stable state. The release enters a beta period. This could be considered a phase of Waterfall; however,I see it as part of the verification phase. During Beta testing, the release is packaged and given to a specific subset of customers to try in their environment.

The Maintenance Phase

Once everybody agreed that the product is ready, it is released, or deemed Generally Available (GA). Once the product is "out the door," the release is in the Maintenance phase, as shown in Figure 1-8. Sustaining engineers or developers address any defects discovered by customers, and business people decide whether another release of the product is needed.

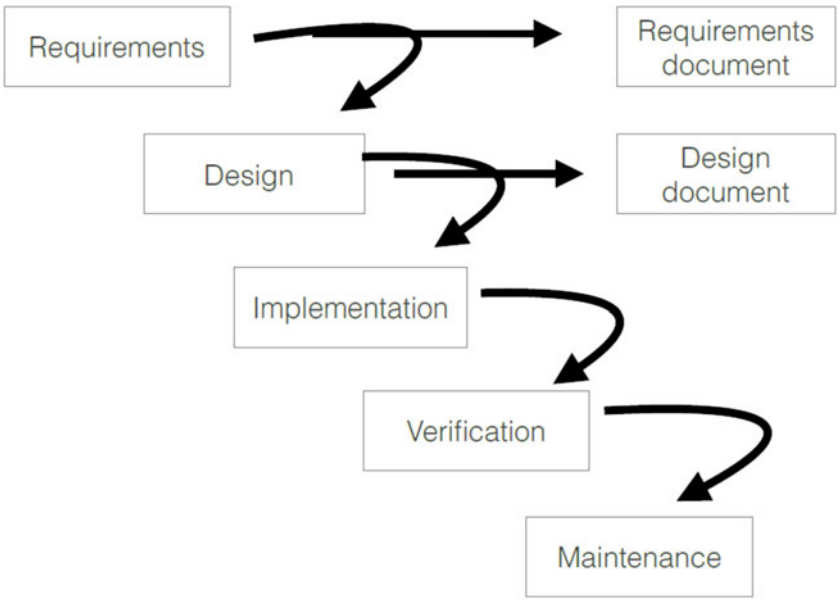

Figure 1-8. The Waterfall Maintenance phase

Makes sense, right? At first glance, Waterfall looks like a sound way to produce quality software. Truth be told, I was involved in the creation of some best-of-breed products using Waterfall.

However, there are some major flaws with Waterfall that are severely problematic here in the application-driven economy of the twenty-first century. If requirements change in a Waterfall project after the requirements phase, everything has to stop and the new requirements need to be evaluated. This causes a domino effect where all of the other phases get "pushed out." This adds cost to the project, and it makes the GA date a whole lot harder to hit. As you can probably imagine, management would not be happy about this turn of events. We now live and work in a world where requirements change rapidly, so there is ample opportunity for requirements to change quite a bit during a twelve or eighteen month release.

You may be asking yourself "Why does Waterfall take so long? Twelve to eighteen months is a long time." One of the biggest reasons I can think of is that customers were used to that cadence. They would try to get as many requirements as possible into the next release because it took a long time to get releases out. That was a big pile of requirements that may or may not be relevant when the functionality is delivered.

It is also problematic that the Quality Assurance folks do not get their hands on anything until the code is finished. Look, I like developers. I really do, but everybody needs to understand that every new line of code is a potential defect. I am not accusing anybody of writing bad code. That being said, we are

all human. Mistakes are made. Couple that with the fact that teams are working on incredibly complex products, and you are going to have some bugs. The earlier you find a bug in the development process, the easier it is to fix. It also costs a lot less to fix. In Waterfall, the bugs did not show up until the very end of the project, in the verification phase. Think about that for a second. How do you predict how many defects were going to be found? How do you plan for that?

In my experience, the verification phase always threatened to push out the GA date because of the amount of time required to fix all of the defects. This resulted in a stress-filled mad scramble to make the GA date. The quality assurance engineers and developers were also two separate teams. This led to an "us versus them" attitude——especially when stress levels were high.

Most importantly, customers and stakeholders did not get to see the new product until Beta or GA. The customer would make their requirements known, and a year later, we would produce a shiny, new thing.

However, how do we know we built the *correct* shiny, new thing?

Allow me to give you an example. I play bass guitar at my church. I am by no means anywhere near a musician, but I can play in the correct key, most of the time. Figure 1-9 shows the instrument that I play is an old Peavey Foundation four-string rig. It stays in tune, and fits me well.

Figure 1-9. My bass guitar is not flashy; however, I can move furniture when I hit an open E string

I found out that my daughter Bailey was going to buy me a new bass guitar for Christmas, and I stopped her. I was touched by the fact that she would put that much thought into a present for me, but there is no way she would be able to buy an instrument that I would be happy with. I am very particular about how my bass plays. The action, the feel of the fretboard, how the pickups are situated on the body. There is no way I can tell if I am going to like a particular bass guitar until I play it.

Same thing with our customers. The team tries to build something that satisfied what was asked for, but more often than not, this was not the case. Sure, we had a Beta period, but that was to make sure the software worked, not to change functionality.

Waterfall development was impacted by changing requirements, unrealistic deadlines, and bad estimating. The biggest impact was on the people. Notice I said *people*, not *resources*. A resource is something you use up, like toilet paper. People are amazing and talented. Waterfall caused low morale, burnout, and even health problems. Something had to be done…

Enter the Agile Manifesto

Personally, I am not overly comfortable with the word *manifesto*. Since I am a child of the Cold War, whenever somebody starts throwing the word *manifesto* around I start thinking of the great workers' paradise that was the former Soviet Union…or the Gulag.

Karl Marx and Friedrich Engels wrote *The Communist Manifesto* to declare their thoughts on what the perfect society would look like. It was a document that changed the world, but it was not the only one. As I see it, a manifesto is a proclamation of ideals and intentions. The *Declaration of Independence* was a manifesto. President Kennedy's *Land a Man on the Moon* speech is another example.

When I was a teenager, I wanted nothing more than to be a monster. No, not like Godzilla or Frankenstein. I was the kid who was picked on. As I saw it muscles and strength were my armor, and I desperately wanted them both. In my mind, a person who could rip a door off the hinges and then have to turn sideways to fit though the door was automatically respected by his peers and feared by bullies. There was a secret… a way to this muscular Nirvana, and I was going to find it.

Mind you, these were the days when the internet only existed in Al Gore's imagination, and training information was nowhere to be found. As a result, my friends and I wasted countless hours flailing away on our plastic Sears and Roebuck weight sets while dumping our hard-earned paper route money into worthless supplements.

Then it happened. I do not know if it was divine intervention, serendipity, or just dumb luck, but a friend's uncle took us under his wing and showed us the secret. We took what he gave us and made a manifesto, of sorts. We wrote it out and taped to the wall of his garage. Right in front of the squat rack.

- **Lift heavy**. Challenge yourself to set a personal record every workout in something.
- **Never miss a repetition**. We were introduced to something called rest/pause training, which I will detail later.
- **You grow when you rest**. Working out for two hours a day, every day will make you small and weak.
- **Eat real food**. A lot of real food. No supplement or drug in the world can take the place of actual food.

As I said before, a manifesto is a public declaration. It is a statement of how we intend to carry out what we do from this point forward. It may not have changed the world, but it changed my world.

In 2001, a bunch of really smart people got together at a ski resort. As I understand things, most of them were lean, or extreme programming (XP) pundits who were coming together to talk about lightweight methodologies. They came up with what we now know as the Agile Manifesto.

Individuals and Interactions over Processes and Tools

As the old joke goes—if you put a bunch of monkeys in a room with typewriters, eventually….by luck you will end up with the complete works of Shakespeare. Supposedly, this is a mathematical possibility…

I doubt it. Mathematically possible or not…

Same idea here. A super well-defined process, the top-of-the-line tools, the best facility in which to work, catered lunches every day, a ping pong table… none of it matters if the team can't work together.

However, if any of my management chain is reading this, catered lunches are a great idea. Just sayin'.

A better bet would be to throw the team in a room and let them figure out the best way to do the job. The idea is that the team needs to talk to each other. When I say talk to each other, I mean everybody talks to each other. I have had the pleasure of working in the software business for over 20 years. That means that I have taken part in many meetings. In those meetings, I have

heard a lot of talking by the senior developers, architects, development managers, directors, and so on. Not a whole lot of input from the junior staff, the QA team, or User Experience engineers. My guess is that they were afraid to speak out of turn because they felt they would be dismissed or made to feel less than adequate.

It is much healthier to engage in collaborative team meetings. The idea is to get everybody on the team to talk to each other, and to understand the perspective they are coming from. For example, I have always said that I could say the dumbest thing you ever heard in a meeting, but still provide valuable input.

That is, what I say is not at all that valuable in itself, but it lets you see where I am coming from. In other words, you understand my perspective, and have a pretty good idea of what my thought process looks like. It is the interaction between people that builds great software, solves problems, and delivers value. People need to be able to communicate and work together. If you do not get that right, you are not going to be successful.

Tools and process can still provide value, but neither is the magic bullet. Take the smartphone, for example. For some reason, the more connected we seem to be in the twenty-first century, the less connected we really are. We seem to want to hide behind our phones and not interact with the person standing next to us. When my daughter and I are at the mall and she sees somebody she does not want to talk to, she pulls out her phone and starts texting or tweeting or whatever the cool kids do these days. I do not understand it. If you leave me by myself at the mall for a half hour, I will have spoken to at least three people and know everything about them. I even make it a point to try to strike up a conversation with folks when they are in the elevator with me. Sometimes, I think I scare them. I guess that whole monster thing backfires on me in this case. Most of the time, folks act somewhat pleasantly surprised that another inhabitant of planet Earth is acknowledging their existence.

I guess I am a people person.

Working Software over Comprehensive Documentation

If you are having problems sleeping, start reading some technical documentation. Think about it—when do you read the documentation that comes with something you buy? Is the first thing you do when you buy a car is to find a comfortable chair and read the owner's manual? I don't even know where the owner's manual for my car currently is.

Most of the really good developers that I know are detail-driven and heavily steeped in process. A developer benefits from these characteristics, no question about it, but it also means they can be easily distracted from what they should be focused on:

Writing software that produces customer value

Teams need to consider the user's perspective when building software. Remember, the goal is to build something that customers actually want.

Documentation is still needed, just not on a grand scale. When generating documentation becomes a priority over generating value, things are out of whack. I am not saying that we stop generating documentation. The key is to have just enough documentation to support the working software. Notice that it says working software. This means zero defects. When bugs are found, they are addressed as soon as possible—even if it means the team stops working on new features to do it.

Customer Collaboration over Contract Negotiation

Customers use our software in ways we never thought of to solve problems we never heard of. The truth of the matter is that developers are not users. The scrum team needs to collaborate with the customer to get a solid understanding of what they are looking for. In order for a team to create software that customers want, you kind of need to know what they, you know … want.

A contract does not take the place of communication. I have been a part of projects that management loved but that never saw the light of day in the datacenter because the people who actually do the job did not see the value in it. Not only must the team collaborate with the customer, they must collaborate with the folks who are "closest to the glass," the people who use the product.

Remember, requirements change rapidly. The customer may not know what they want at first. They will change their minds. They may not do a good job of communicating. Constant collaboration allows the team to overcome these obstacles and deliver something truly valuable to the customer. It allows the team to focus continuously on the end goal, and not anything else.

Responding to Change Over Following a Plan

Change is disruptive. Change is messy. Avoid change at all cost. That is how change was viewed before the Agile Manifesto. That was then. This is now. The application economy is disrupting how customers go about their business, and how that changes what they require from the products they buy. These days, requirements change so quickly that it is foolish to try to plan out a large-scale release up front.

Agile teams need to embrace change. This means that they need to shift how work is done. We know change is going to happen. We should be expecting it. Part of our release plan should be to adapt to change. Instead of saying "Look at what I built for you—Isn't it cool?" we should be thinking about how

to deliver small slices of what we plan to build at regular intervals. This way customers can get their hands on what we are building. They get to try the new functionality and give us feedback.

And expect disruptive change to come from the customer feedback we get.

That means that we do not plan 10 months ahead in intricate detail. A team's release plan and Backlog priority should be constantly changing and evolving due to changing requirements and feedback.

The power is in the ability to pivot and adapt quickly. This ensures that what the team builds is what the customer wants. Even if it isn't what was originally planned.

Wait, There's More

In addition to the manifesto, the authors defined 12 principles. Most folks do not refer to the guiding principles when they talk about the Agile Manifesto. I am not sure why. I think the guiding principles are just as important as what is contained in the Agile manifesto itself, if not more. These principles are simple enough to understand, they but also have a deeper complexity that brings out the deeper meaning of Agile.

1. Our highest priority is to satisfy the customer early and continuously deliver valuable software

Back when I was working in customer support, we used the phrase "delight the customer." Think about that for a second. When you are satisfied with something, it is a good thing. There is nothing wrong with satisfied, but when you are delighted, that is taking it to a completely new level. If I am finished with a big meal at a fine dining establishment, I am probably satisfied. I will be smiling, patting my belly, and generally happy with what I just ate. If the meal delighted me, I cannot wait to tell all my friends about the awesome meal I had at this establishment. I am on social media as fast as possible, telling everybody about my experience. This became a mantra of sorts——do not just satisfy the customer, delight them. That is what we should be all about——delighting the customer—Not delighting management. Delighting our customers. If the teams focus on anything else but the customer, then the focus is in the wrong place. The goal is not just to produce something that the customer wants, but something that the customer cannot wait to get their hands on.

This requires developers to change the way they work. It is necessary to rework the way code is written. It may require the team to change the way they work together. This can be a struggle, especially if not everyone on the

team is comfortable with this idea. The team needs to understand that it is better to get something wrong upfront and have time to pivot and deliver what the customer really wants.

Think of it like a wedding cake. Let us say that you are paying for a wedding.

Yes, that is extremely frightening. Trust me, I have two daughters.

I think everybody would agree that next to the bride's dress, the wedding cake is the most important decision to be made. Knowing this, as the person buying the cake, you want it to be perfect. I am sure that the pastry chef making the cake wants to delight you.

However, if a huge, seven-tier cake shows up the day of the wedding and the bride does not like it, there is a good chance we have a living, breathing Bridezilla on our hands.

Trust me; nobody wants that, so we have a tasting. We get to taste the cake and see what it is going to look like. I would not expect to see the huge cake here. I would expect a Minimally Viable Product. A smaller version of the finished product that stands on its own. We get to taste the cake, icing, filling, and whatever. We could also possibly see some of the decorating techniques that are planned, but we are not seeing the finished cake. This way, we can give feedback and let the pastry chef know what we want.

Maybe we wanted strawberry filling between the layers, but now we do not like the idea, or we like flowers instead of birds, or fondant instead of butter cream icing.

The idea is that we get an idea of what the final product will be by working with a small prototype. We can then give feedback and get exactly what we want. In the wedding cake example, we might only do this one or twice. When developing software, the team delivers early and frequently so that there is plenty of opportunity for feedback from the people who are going to use the software. That ensures that what you are building is not the wrong thing but the right thing.

Which brings us to the most important point. In the end, all the customer cares about is that what you have produced is something valuable to them. That is what the team needs to focus on.

2. Welcome changing requirements even late in development

Agile is about adapting to change, not planning everything out up front. If you have not noticed yet, I have been talking about change quite a bit. Change is the reason teams need to change the way work is done, but it is more than that. Customers cannot make their requirements known, *because those requirements are constantly changing.*

Agile teams work in a manner that expects and embraces change. This way, customer requirements can be satisfied—even if they change late in the project.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale

I remember when I challenged my team to deliver what we were building in small increments. "We can't deliver all of that functionality in that short amount of time." they said.

"What if you work on some of the functionality and mock up the rest?" I suggested.

The look on their faces was priceless. It was as if I suggested that we all cut off a finger and try to sell them. Seriously, they were horrified.

It is human nature to not want to show something off until it's done. But it does not make sense to hold stuff back until we think it is, well… good. A team needs feedback as early in the project as possible, and then at regular intervals. The absolute best way for a customer to give useful feedback is to try out what has been built. A team can produce slideshows, technical diagrams, and even demos of the functionality they plan to produce, but nothing is better than working software. A team works in iterations, or *Sprints,* that last from two to four weeks. The goal of the Sprint is to produce a minimally viable product (MVP) that the customer can take "for a test drive." After playing with the software, the customer can provide the team with the feedback they so desperately crave. An added bonus is that the status of the project is as transparent as can be. The customer knows exactly what the team is producing as well as how long it is taking.

Speaking of that, a team should strive to make Sprints as short as possible. That makes sense if you think about it. The shorter the Sprint, the more feedback the customer can provide.

4. Business people and developers must work together daily throughout the project

The idea behind a software project is to satisfy the business. In other words, to make money. It is easy for a team to focus on the wrong thing. When you keep the team focused on the business, they have the proper perspective. The team learns much more about the business this way than by going over lists of requirements. The team needs to be encouraged to interact with the

customer as much as possible. It may not be possible for the team and customer to interact every day, but contact should be as often as possible.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done

The team is the most important entity. It may not be possible to build a "dream team," but to be honest it really does not matter as much as you think. That being said, management cannot throw a bunch of junior programmers together and expect miracles to happen. The key is to form a team that can work together, and then let them do the work.

In the Waterfall days, the development manager would tell the team what to do and how to do it. Customers took every opportunity to try to get the team to put the functionality they wanted into the product. Salespeople would bother the team to drop everything and provide functionality that would help to close a sale. Teams were also asked to provide demos, help support, and a bunch of stuff I am forgetting.

An Agile team is insulated from all of this. Instead of a development manager, the team has a Product Owner that is responsible for talking to customers, understanding how they work and what value they would gain from the project, and building a prioritized Backlog of work. The team uses the Backlog to build what the customer wants. The Scrum Master (another Agile role) is responsible for protecting the team. All of the distractions are gone, allowing the team to focus.

6. The most efficient and effective method of conveying information to and within a Development Team is face-to-face conversation

Face-to-face communication is most effective because all facets of communication are in play. You have facial expressions, tone, body language, and the actual words being spoken.

One of my favorite coaching experiences was when the mother of a player marched up to the head coach and me. I could tell by the look on her face that she was upset about something. Actually, mad as a wet hen is probably a better description. She got uncomfortably close to the head coach and said "I didn't appreciate the tone of your email." She then pivoted and walked away.

"How do you get tone from an email?" said the head coach with a quizzical look on his face…

Communication is vitally important. Agile won't work without it, so we should be striving to use the most effective method possible. If you try to communicate using email or some type of Instant Messenger, you are losing one or more facets of the communicative process.

And it's faster. If we were communicating via email, you would need to wait until I read the original email and respond to find out if I even understand what you are trying to say, let alone my feelings on the subject. Face to face, that happens immediately. As a result, understanding flows easily.

7. Working software is the primary measure of progress

Working software needs to be where the team is focused. Not on documentation, or design, or anything else. Development Teams seem to want to focus on completing development. Completed development and working software are not necessarily the same thing.

In other words, software is not done if it compiles or assembles. It has to "work," meaning it can be tested and is demonstrable.

Let's go back to the wedding cake example. If the pastry chef shows up at the cake tasting with a recipe in his hand and tells you that he's perfected it for your wedding but has no cake for you to test, would you be happy?

I think not…

To be successful, the pastry chef needs to focus on tasty cake. The team needs to focus on working software.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

The mad rush to get projects to the GA date required long hours. The teams that I was involved with pulled together and got the work done, but it was not fun. I am sure that the pace wasn't sustainable for more than a few weeks.

Look at it this way. The fastest marathon runners in the world run the 26.2 mile course at a six-minute mile pace. That pace is sustainable for them, but it is far from an all-out sprint. Nobody in their right mind would try to sprint for the entire length of a marathon.

Neither should a Development Team.

When I was in basic training, the drill sergeant was fond of saying that if we were supposed to have a family, it would have been issued to us and grounded at the right side of our foot locker. This should not be the prevailing attitude of the people you work with. Work-life balance is important. A sustainable pace supports a healthy work-life balance.

9. Continuous attention to technical excellence and good design enhances agility

Clean code is easier to work with than spaghetti code. That should be obvious. I doubt that any person who ever worked as a developer woke up one day and said "Today I will write code that is complex, convoluted, and hard to understand." It happens because of the pressure required to get projects complete and make dates. An Agile team needs to be constantly mindful of the code they write. Once the code becomes a mess, you lose the opportunity to pivot quickly and respond to feedback. In other words, you lose agility.

The same with design. Why go off and spend a great deal of time coming up with a complex design for the entire project when you are expecting change? Design must be clean, efficient, and done in a way that embraces change. I like to say that coding and design must accept change *gracefully*.

10. Simplicity—the art of maximizing the amount of work *not* done—is essential

This seems counterintuitive. After all, a Development Team exists primarily to write software.

Yes, that is the idea. The team should be all about building awesome stuff——just as long as they are building the *right stuff*.

This requires development to reframe the way they think about writing code. If a developer writes a simple function, they will assume they need to look at all possible scenarios and do all kinds of work that really isn't necessary.

"Well, this might happen, so I better build something into the function for it."

"While I'm at it, I might as well call this other routine to make sure the data is good… And I guess I should prepare for this other thing as well."

An Agile team should create the function as simply as possible. Get in, do the work as simply as possible, and get out. I'm being simplistic here (while discussing simplicity, no less), but this is the way teams need to look at development. All of the other stuff you think you "might" need is a waste of time. If you need to add it in the future, add it then. I've heard it said that the majority of features built are hardly ever used. Complexity leads to unreliability. Agile teams strive for simplicity by building only what is needed at the time.

11. The best architectures, requirements, and designs emerge from self-organizing teams

I coached cross-country for 5 years when my daughters ran. The sport of cross-country is exactly what it sounds like. You run in the woods. My coaching style would normally consist of me showing the runners the course and saying "get after it." I would then stand at the finish line with a stopwatch and wait for them.

Parents would always ask me if I was going to run with the kids or give them pointers on how to race. My answer was always the same:

"They will figure it out themselves."

The truth is that I would do more to screw things up than fix them if I got too involved. Running is simple; we all can do it. Racing requires dedication and desire. Somebody like me yelling at you to run faster isn't the proper motivation. Wanting to beat the kid in the different-colored shirt is. I always found it amazing that after the first race the team stopped walking and started running—hard. The proverbial light bulb went on.

It's the same thing with an Agile team. They don't need a development manager who tells them how to do everything. Just like the runners, the team figures it out as they go. Instead of trying to design everything up front, the team allows the design to emerge as they are building the functionality. Give them a goal and get out of the way. They will figure out the best way to get there.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts

> Agile is about inspect and adapt with a focus on experimentation.
>
> —Bob Carpenter, *CSM, ICP-ACC, CSP*

Agile is all about continuous improvement. An Agile team inspects everything that it does and routinely makes changes in an effort to try to get better. Doing a post-mortem after a project is great, but the project is over. That ship has sailed, and the lessons learned can only be applied to a future project. By contrast, an Agile team takes the time to do regular retrospectives *during* the project. In these retrospectives, the team identifies ways to improve and tries to incorporate them into the way they work. The team gets to try new things, keep what works, and throw out what doesn't. By focusing on constant improvement, the team can't help but *improve*.

That constant improvement creates high-performing teams.

This chapter provided an overview of Waterfall software development, the Agile manifesto, and the twelve principles of Agile software. The next chapter will examine Scrum, a lightweight framework that promotes iterative development as described in the Agile Manifesto.

CHAPTER

2

The Scrum Framework

There are around a dozen Agile methods being used today. Of them, the Scrum framework is the most popular. Agility comes from increasing feedback loops and responding to that feedback. The Scrum framework encourages short feedback loops, lowers risk, and allows return on investment (ROI) to be seen sooner.

Roles, Vision, and Backlog

Scrum requires three roles that you may not be familiar with—product owner (PO), Scrum Master, and team member. I will go into greater detail about these roles in the next chapter.

Some organizations have an additional role that is outside of the Scrum framework, but very important to the team's success, called the product manager (PM). The PM is responsible for defining a vision of what this product is going to look like. The PM is responsible for what is referred to as the *product roadmap*. In other words, what will this product look like in five years, three years, one year from now? How will it play in the market? The vision is what funds the project and gives everybody a reason to get up in the morning and come to work.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_2

That vision gets turned into a Backlog. Remember the requirements document from Waterfall? The Backlog is like that, except it is constantly being defined and prioritized. It is a list of stuff that needs to get done that will turn the vision into reality.

Literally anything can go into the Backlog. Stuff like defects, technical debt, enhancement requests, even risks. Functional and nonfunctional requirements. The only catch is that Backlog items must be expressed in business (nontechnical) terms.

Stories and Epics

The items in the Backlog are referred to as *user stories*. Written from the user perspective, stories convey small chunks of functionality or value for the team to work on. A story is a simple description written in a nontechnical way. The idea is to shift the focus from writing about features to talking about them.

An *epic* is a great big user story that needs to be broken down, or a collection of user stories. Epics are an easy way to organize the Backlog in a clear way without having to get too deep into complexity. Think of this in the terms of music. You can sort your music by genre, artist, album, and song. Each level gets a bit more specific and complex. You can look and see that you have a lot of music in the Heavy Metal genre, and then break it down and see how many songs are there by album and/or artist. If you want to group your music by a certain theme, you can. Same thing with the Backlog using epics.

The Product Owner Owns the Backlog

The *product owner* (PO) is responsible for the Backlog. I like to say that the PO *lives* in the Backlog. The PO understands both the business and the product the team is to create. They know what the product is supposed to look like, what it is supposed to do, and most importantly what value it will bring to the customers. Having a well-defined Backlog enables the PO to encourage the team to create something of value. It is important to note that when items are first placed in the Backlog they can be very general. As the epics are broken down into stories and the stories are refined, they get smaller and more detailed. An image I like to use is stories "rising" in the Backlog, as shown in Figure 2-1. By the time a story gets to the top of the heap (prioritized the highest and ready to be worked on) it should be well defined and ready to be worked on by the Scrum team.

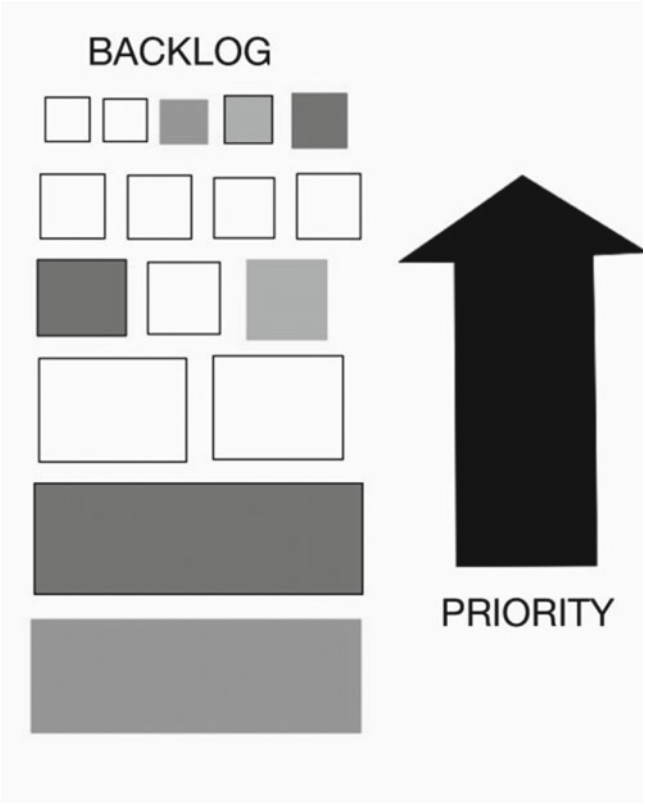

Figure 2-1. A healthy Backlog

It is important to understand that we don't expect to know all of the requirements up front. In other words, a PO is not expected to build a perfect Backlog before the team starts coding. The Backlog will grow and evolve as the Scrum process plays out and requirements change and/or become clearer.

The PO adds stories to the Backlog and prioritizes them according to business value. That business value can change rapidly, so the PO should be constantly prioritizing the Backlog. The stories that provide the highest business value at the current time should be at the top of the list. The PO understands how customers use software as well as what value they get from it. They are concerned with what and why, not how. It's up to the team to figure out how to deliver what the PO is asking for. The product owner should be able to not only be able to clearly articulate what is wanted, but why it is important. People don't really invest in what you want done, but when you tell them why—the purpose, the cause… they connect and get excited about doing the work.

Stakeholders (both internal and external) want a say in how the product is built. External stakeholders are customers, aka the people who give us money for what we build. An example of an internal customer would be product support. An internal stakeholder has a keen interest in what the team is building but is part of the same company. The PO acts as a buffer between all of these people and the Scrum team. For example, if the legal department requires the team to publish an end user license agreement for some third-party software they are using, they don't go directly to the team. They go to the PO, who will put that requirement into the Backlog and make sure it is properly prioritized. That EULA (End User License Agreement) may not have to get done right now, but it needs to be done before the product is released. This allows the team to focus on the task at hand and not be bothered by external influences. The team is shielded because the PO takes their requests and creates a Backlog of user stories.

The Sprint

Speaking of the team, a Scrum team is a cross-functional team that works in a fashion that is very different from the Waterfall days. The team commits to achieve a small number of goals in a short period of time. The team does whatever it takes to achieve those goals. When I say "whatever it takes," I don't mean keeping a sleeping bag at their desk and never seeing their families. I am referring to the fact that a team member does what needs to be done regardless of role. For example, a developer may do some QA work to get something done and a goal realized, or vice versa.

The team will take the product Backlog, and with the help of the Scrum Master pull stories into a Sprint Backlog during the Sprint planning meeting. At this meeting, the PO discusses what he or she would like to see accomplished in the Sprint, and the team commits to getting a certain number of stories done to achieve that Sprint goal. It is important to note that once the Sprint Backlog is set, nobody can change it.

The word "commit" has been frowned upon lately, but I think it is too powerful not to use. What I find powerful is that the team no longer works to meet unrealistic demands put forth by management. They still have the pressure to complete something by Sprint end, but only what they have committed to. Look at it this way: If I tell you I am going to do something, I'm invested in it. It is not the same if *you* tell me to do it.

A Sprint is a 2–4 week iteration in which the team works to produce a potentially shippable product increment (the *minimally viable product*) as shown in Figure 2-2.

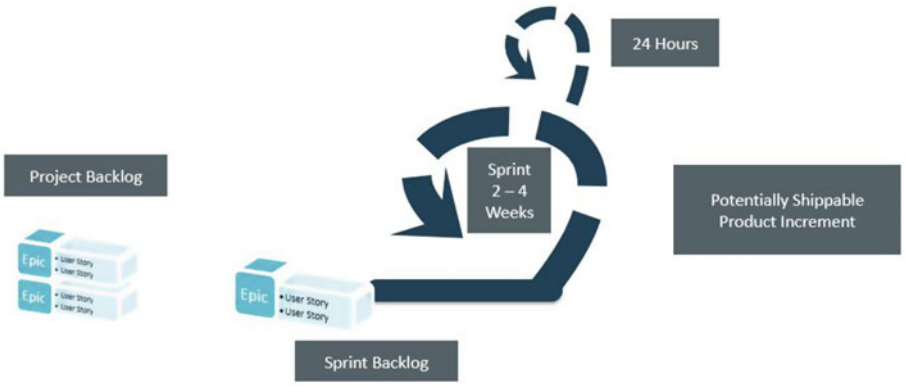

Figure 2-2. A Scrum iteration

Why 2–4 weeks? Because that is a time period that is easy to control. Most human beings are really good at putting things off until they absolutely have to be done. Limiting the team's "time box" to 24 weeks forces them to get after the task at hand. There is no time to waste. At the end of the Sprint, the team holds a Sprint review meeting where the team shows the stakeholders what it has built and asks for feedback. The team then takes that feedback, adjusts accordingly, and iterates again with another 2–4 week Sprint. Sprinting over and over again and reacting to the feedback given about what is produced is extremely powerful.

I'm painting with a broad brush here, but think of a Sprint as a little release contained in a 2–4 week window. The team is continuously releasing working software to the stakeholders at every Sprint boundary and allowing the stakeholders to give their opinion of it. The key task is to take that feedback and use it to create something of value to the customer.

While working the Sprint, the team iterates daily (iterating within an iteration… I'm getting dizzy here) by having a *standup*, or *daily Scrum* meeting. The standup meeting is not a status meeting, or a development meeting, or a staff meeting. It is for the team to tell each other about what they got done the previous day, what they plan on doing today, and if they need any help with anything.

This is so that the team can collaborate and work efficiently, and to handle problems as quickly as possible. I should also note that every part of the functionality needs to be complete by the end of the Sprint. That includes all QA, documentation, and so on. At the end of each Sprint, the team needs to provide a "potentially shippable product."

Speaking of done, how does a team know they are done? By "done" I mean done. Not just code complete. Not done except for a few automated tests that need written. Done… complete… ready to ship.

Working to the Definition of Done

The team develops, designs, and agrees on a *definition of done* (DoD). The DoD is a checklist of things that need to be completed before the story is considered "done." It is a very simple concept but probably the most important thing a Scrum team can do to achieve success. When a DoD is defined, both the team and Product Owner can be comfortable with the fact that everything on that list has been satisfied. It should contain milestones like the code being complete and checked into source management. Automated QA scripts have been written, and manual QA is complete. Documentation has been published, and so on. The team works to satisfy the DoD and the user story acceptance criteria. Acceptance criteria are written as part of the user story. They are an agreement between the PO and team as to what the PO needs to see to accept the story. I like to explain it this way:

Let's say that I want a cup of coffee, and I ask you to get me one.

With creamer.

At this point, I really don't care how you actually get the coffee. I just want a cup. In Agile terms, I've just defined a story.

So it's now your job to take this story out of the Backlog and make it happen, but how do you know when you have satisfied my request? How do you know that you have made me happy?

In Agile terms, this is why we have a definition of done and acceptance criteria. The DoD drives Scrum team responsibility—who has the skills to get this thing done, and what tasks need to be completed for us as a team to say we are "done"? The DoD is *not* the acceptance criteria attached to a story.

In our example, my acceptance criteria is that a hot cup of coffee shows up on my desk. I don't care if you went to the kitchen on the third floor, or to the cafeteria downstairs, or got in your car and drove to Starbucks—or Dunkin Donuts. I just care about the end result; that is what *acceptance criteria* means. Technically, the acceptance criteria will cover all of the products owner's conditions of satisfaction.

My definition of "done" would look something like this:

- Cup is filled with coffee.
- Coffee is hot.
- Coffee is fresh.

- Cream has been added.
- Coffee is delivered.

Now, you might be thinking, "Where did I get the cup?" Details like that would be covered in tasks that you then add to the story.

Having a properly defined DoD enables the Scrum team to become efficient—the team works to satisfy the DoD. This enables both the team and product owner to understand that a story is done. Once the story is done, the PO can determine if the story satisfies the acceptance criteria: does it deliver the promised functionality? In fact, the first implied step of the DoD is that the story meets the acceptance criteria. Getting back to my example, I can trust that my coffee will be delivered fresh and hot. I don't have to ask anything of you, and you can rest assured that I will be happy with what you have delivered to me because my acceptance criteria (what I expect) is clearly defined.

There is no longer a development manager that assigns work to the team. The team decides what it wants to work on. They are self-driving. The idea is that the people developing the software are the experts. They know how to best build what is needed. While working, the team is as transparent as possible so that everybody is aware of what is going on, and things can be easily corrected if problems arise.

At the end of a Sprint, the team, PO, and product manager hold a sprint review meeting. They invite the stakeholders, show them the functionality they completed during the Sprint (preferably by doing a demo), and get feedback about what the team has done. The stakeholders are also invited to install the minimally viable product (if possible) and take it for a "test drive." By doing this, the team is getting feedback throughout the development process, not just at the end. It enables the team to react to the stakeholder feedback and really deliver value. If the stakeholders don't like something, the team can react quickly make changes. If the stakeholders love the product, they might want it to be released right away.

After the Sprint review meeting, the team will hold a retrospective. This is effectively a post-mortem about the Sprint that was just completed, covering how the team did, and what they need to do to get better the next Sprint.

And then everybody does it all again in the next iteration. Where has the Scrum Master been during this whole process?

The Scrum Master's job is to drive the Agile processes. The Scrum Master makes sure all the team members live out the values and practices of Agile. The Scrum Master sets up and holds the meetings, removes obstacles, breaks up fights, moderates disagreements, and buys the donuts.

OK… seriously, the Scrum Master does everything possible to facilitate productivity.

A Quick Recap

So as you can see, by iterating often, getting tons of feedback on each iteration, and taking action because of that feedback, we are setting up an environment where we are delivering features that the people who use our software really want. Working in this fashion allows us to pivot quickly and react to changing requirements and customer demands. Scrum allows the business to drive development, as the PO should have the business in mind as the Backlog is prioritized. However, Scrum teams are also allowed to change their minds as they learn new things. It is a new way of working that embraces the Agile Manifesto.

Allow me to recap:

- The product owner creates a refined, prioritized Backlog that consists of user stories.
- The Scrum team works from the Backlog. They commit to how much work they can do in a Sprint, and move those stories into a Sprint Backlog.
- The team Sprints, or iterates every 2–4 weeks. The PO and team agree to a goal every Sprint, and the team works to achieve that goal by completing the stories in the Sprint Backlog.
- Every 24 hours, the team gets together to discuss their progress in a stand-up meeting. They answer three questions: What did I do yesterday? What do I plan to do today? What is blocking me?
- Anything blocking the team is aggressively removed by the Scrum Master.
- At the end of each Sprint, the team hosts a Sprint review meeting where they show the stakeholders what they have produced and collect feedback.
- Before starting the next Sprint, the team holds a retrospective where they reflect on the previous Sprint and ways to improve.

Scrum is a very simple framework that is very difficult to apply. It requires a change in culture that requires a new way of thinking. In this chapter, we took a detailed look at the Scrum framework. In the next chapter, we take a closer look at the Scrum roles and how they impact the framework.

CHAPTER

3

Scrum Roles

Embracing Agile is not a process change; it is a culture change. Believe me, the culture change stuff is what is going to keep you up at night. Making Agile work will require people to embrace new roles and to change the way they do their day-to-day work. Before I dig into the Agile roles, now is a good time to detail what I believe to be the three pillars that everybody who works in an Agile environment needs to be aware of. The three pillars of Scrum are transparency, inspection, and adaptation.

The Three Pillars of Scrum

I was coaching some folks from the support organization. I kept bringing the conversation back to the fact that everything really does get better if everybody is operating in a transparent fashion. One of the people in management asked "What is transparency?"

I was taken aback by this question. I mean, isn't it self-explanatory? When you are transparent, you are…well, transparent.

Maybe this isn't so self-explanatory after all.

Transparency means that *everything* having to do with the project is out in the open for all to see. I like to describe it as oversharing. All of the team's success and failure is out in the open for all to see. The team is operating in a transparent manner and sharing information; the product owner is also being transparent and sharing. The Scrum Master posts all relevant team information on information radiators so that stakeholders can easily find out how the Sprint is progressing. A *radiator* is a display (or dashboard) that is put in

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_3

a high-traffic area. Instead of hiding information, a Scrum team broadcasts everything about what they are up to. This is something that people may have an issue with at first. Human beings have a tendency to want to hide things. How many times did you hear your parents say "Mind your own business" or "Keep it to yourself" when you were growing up? Most of us did not grow up in an environment that encouraged us to be open and transparent—especially about failure.

Development teams naturally want to hide problems until the last possible moment. In the Waterfall days teams would hide issues from the development manager, hoping to fix them before they came to light. Scrum calls for work to be done out in the open so that problems can be addressed as soon as possible.

My brother was coaching offensive line at the high school that we both graduated from. I went and lifted weights with the team a few times, but I wouldn't consider myself a part of the team or anything. That's why I was so excited when my brother gave me a team "ironman" shirt. Each member of the team that was participating in the off season workouts got one. On the front it said "Viking Iron-men" around a picture of the team mascot—a big, muscular Viking holding a loaded barbell. On the back were the 10 "commandments," basically the core values of the program. I was psyched about the shirt. I wore it all the time, However, I'm not telling you about this because I liked the shirt so much.

I'm telling you because of the way some folks reacted to it.

On seeing my new shirt, one of my brothers–in-law said, "I don't agree with number six."

Number six said "You have to lose to win."

On the surface, it doesn't make sense. Our culture loves a winner, so there is a huge premium put on being right. It's almost a giant game show:

"I'll take being right all the time for 500!"

Because of this, there is a real, tangible fear of being wrong. If we walk around with a fear of being wrong, we will never grow. I've always said that if you show me someone who has never failed, you are really showing me someone who has failed to learn. Focus on the positive outcome of failure. No, nobody likes to fail. I prefer to think of something not working out as a wonderful way to learn what does and does not work. A Scrum team needs to be willing to try new things—even if that means they will fail at it.

You Need to Lose to Win

"You have to lose to win" was a training motto for the team weight room. I heard Tony Pharr, one of my training mentors, seen in Figure 3-1, mutter that phrase at least a million times. It became part of my lifting DNA, and rubbed off on my brother as well. The basis was simple. A lifter needs to be constantly tested. Playing it safe will not lead anybody to greatness. The program that we all started all of those years ago in a garage was very simple conceptually, but very hard to implement because it required the lifter to be constantly attempting to set a personal record (PR) in something. The program looked like this:

Week 1: Work up to your best set of 15 repetitions.

Week 2: Work up to your best set of 5 repetitions.

Week 3: Work up to your best set of 3 repetitions.

Week 4: Work up to your best single.

Week 5: Rest.

Then repeat the cycle…

Figure 3-1. This is Tony Pharr bench-pressing more than me

As I said, it's simple but really difficult because you were constantly testing yourself. When you failed, there were two options. Make the failed repetitions up or lower the weight and start again.

Let's say that you were going for 225 pounds for 15 repetitions on the bench press, and you fail at rep 12. You could rack the weight, sit up, and count to ten (I'm talking "one Mississippi, two Mississippi here.") You need to recover enough to have a chance to complete your set, but not fully recover.

When you reach ten, flop back down on the bench and do your best to grind out five repetitions. Yes, we added one. There was no technical reason I can think of as to why we added a rep. We just did it.

Your other option would be to drop the weight, say to 220, and get your 15 reps. The only caveat was that it had to be more than the last time you tested yourself for 15 reps. Talk about continuous improvement!

As you can see, the lifter was always being challenged. There was constant pressure to deliver. Just like the Sprints in Scrum.

A command–and-control culture magnifies the fear of being wrong, or failing. Scrum suggests that you be totally transparent so that you can inspect what you are doing often. If the team fails at something, determine why and adjust accordingly. On the last day of every Sprint, the Scrum team should hold a retrospective—basically a team meeting where the team and Scrum master get together, look at how things went during the Sprint, and then figure out how to get better. Operating in a transparent fashion is a great thing to do; however, if you don't ever take a look at what's going on and determine how to get better, being transparent doesn't add any value.

Having a regular retrospective meeting creates, or at least helps to bring about, a culture of continuous improvement. By continuously looking for ways to improve, you can't help but get better.

Transparency, inspection, and adaptation are referred to as the "three pillars of Scrum." To become great at something, you have to objectively look back at what you have done. Keep what works, and throw out what does not.

Scrum Roles

The Product Owner, Scrum Master, and Scrum team are three roles defined by the Scrum framework. These roles are what unlock the power of Scrum and enable value to be delivered to customers and stakeholders rapidly.

The Product Owner

I've heard the Product Owner (PO) role described in a number of ways. The keeper of the Product Backlog. The customer advocate. *The one wringable neck.*

One thing for sure, the PO role is challenging.

It is the contact point between the Scrum team, the customer, and the business. The Product Owner talks to customers and gathers requirements. They build a backlog of user stories that the Scrum team uses to turn those requirements into working software. The PO also needs to be all about maximizing return on investment (ROI). They needs to constantly think about satisfying both the business and the customer. Every decision a PO makes is weighed against how it affects the business while delivering value to the customers. They have the authority to accept or reject completed work, and can even change the direction of the project at the end of every Sprint. Yeah, that is a lot of responsibility. The PO has influence over the happiness of both the team building the software and the people using it.

Customer Advocate

The Product Owner needs to have expertise on how customers use software as well as what value they get from it. The role sets them up as the single point of contact between the team and all of the stakeholders. By "stakeholder," I mean anybody who is interested in what the team is producing: executives want to know what the return on investment is; support reps want to know how to debug new features. Other teams may want to know how to interface their product to yours.

Stakeholders are very good at making their requests known. Well, maybe that is a stretch. They are good at telling anybody who will listen what they want. What they do not realize is that all they are doing is distracting the people who should be focused on the task at hand. For example, I have a complex jigsaw puzzle and I offer you $100 to put it together for me. The catch is that I only give you 3 hours to do it. I'd bet that you couldn't complete the task if people kept interrupting you with questions that required you to shift your focus from the puzzle.

And customers just love to talk to developers about their requirements. I can remember going to user conferences and seriously thinking that solitary confinement really was not such a bad thing. Seriously, I talked shop from the time I left my hotel in the morning until my head hit the pillow at night. The truth of the matter is that customers desperately want certain features, and don't know how to get them. They need a way to make their needs known in a meaningful way.

That is why the product owner is so important. A PO's interface with the customers and stakeholders is twofold. Talking to the PO is the sure way to get your "ask" into the Backlog.

Customers don't have to try to get functionality in via the back door; they have full access to the front door—*the Product Owner*. Meeting with the PO and ensuring that there is an understanding of the requirements and urgency of need is the best way to get what you want into the product. This also helps to keep the team from being distracted. The Product Owner is the single point of contact for the Scrum team. The team is no longer bothered by any of the stakeholders. Everything is done through the PO. This enables the Scrum team to do what they do best—create valuable software.

Maximize ROI

I've never heard of a customer's IT budget getting bigger. That's why the manifesto states "over contract negotiation." A CIO (or whoever is doing the negotiation) will always say that their budget is shrinking and that they can't pay what is being asked for the product. That's why we need to produce software that the customer can't do without. Stuff they've just got to have. The Product Owner needs to partner with the customers to get an intimate understanding of how they use the software and what they find valuable.

Developers don't work in data centers; they write code. They may have worked in what I would call a customer environment at one time, but now their role is different. They are good at what they do, but they need to understand how the customers use the software in order to create something valuable. As I said before, customers use our software in ways we never thought of to solve problems we never heard of. I can remember learning about a customer using the product I was working on at the time to monitor refrigerators. I was working on a team that supported an outboard automation product. It had the ability to gain access to a variety of hosts and take action on events. Hosts that it could monitor included mainframes, windows servers, and IBM iSeries, but not refrigerators. The customer had a need and figured out a way to do it, and gained value as a result.

The voice of the customer needs to come from, well… the customer. It is the job of the product owner to take that customer voice and keep the team focused on it. The PO must be able to communicate effectively—they are the source of information for the Scrum team. If there are any questions about what is being built, how the requirements are reflected in the story, or even how a customer will use what is being built, the team will expect the PO to have the answers. The PO needs to be able to fill the role of the customer when the team has questions.

The Lord of the Backlog

The Product Owner is responsible for the Product Backlog. Think of the Backlog as a "to-do" list. Daily, I create a to-do list in order to get stuff done. The order of the items on my list change as priorities change throughout the day (mostly because my day almost never turns out as I expect it to), but that doesn't really affect my work very much. As I finish items, I go back and take the next one off the list. At the risk of being simplistic, that is how the Backlog works. A list of requirements is prioritized and refined and eventually turned into deliverables.

The PO may not write every story in the Backlog, but they are responsible for everything in the Backlog. It may be unrealistic to expect them to physically write every story. Team members can pitch in and help; however, the PO needs to be keenly aware of everything that is going into the Backlog. By being aware of each story in the Backlog and keeping their finger on the pulse of the business, the PO can effectively prioritize the Backlog and keep everybody aware of what is coming up next.

The Product Owner is also the person who has the authority to accept or reject user stories when the team is done with them. He decides if the story satisfies his acceptance criteria or not.

Acceptance criteria, as described in Chapter 2, are a list of conditions that must be satisfied before the PO will accept the story. The list is something that both the PO and Scrum team agree to, and it ensures that the objectives of the story are met. A well-written acceptance criteria document will focus the team on the desired outcomes of the story. It is a form of a working contract between the Scrum team and PO. The acceptance criteria list is created by the PO, and then refined and agreed to by the team.

I've heard it said that the Product Owner is responsible for accepting stories into the Backlog and accepting stories when the team completes them. Sort of like cradle-to-grave coverage.

The PO is ultimately responsible for the project, so they must be invested in it. There will always be more demand than capacity. It is the PO's job to ask the right questions to ensure that the Backlog is properly prioritized. A properly prioritized Backlog leads to the team producing the correct level of value that will satisfy as many stakeholders as possible. I will go into further detail later in this book.

To recap, the Product Owner is responsible for building a prioritized Backlog of user stories that the team uses to create value. The stories create clear goals for the team to achieve. If the Scrum team has any questions about a user story, it is the PO who provides the answer. It is the PO who refines the stories with the Scrum team's help and gets them to the point where they can be pulled into a SprintSprint. The PO is the point of contact for customers

and stakeholders. Because of this, the Scrum team is free of distractions and can focus on delivering value each and every SprintSprint.

The Scrum Team

A Scrum team is a bit different from what you may be used to. Because it's defined as a cross-functional self-organizing team, there are not supposed to be any defined roles.

Not supposed to be....

The reality is that there are no defined roles on a Scrum team, but most people come to a team with a defined role. Scrum teams must be able to write, test, document, and deliver working software every Sprint. Defined roles make this very difficult.

Do What It Takes to Achieve the Goal

A basketball team has five players on the court. Each player has a role, but will do whatever is required to win the game. When I coached (Yes, I coached my daughter's basketball teams as well), my pet peeve was rebounding. Rebounding is where a shot is missed, and you go up and get the ball. The team that "cleans the glass" has a distinct advantage. I would tell my players to read the box score of any basketball game, high school, college, or NBA. The team that has the most rebounds is usually the winner. Rebounding is hard work. You need to get position on your opponent, out-jump them, and get the ball. It really is *hard* work, and certainly there is no glamour in it. Could you imagine how things would turn out if the point guard refused to go after a rebound because that was the center's role?

Developers write code, testers test, doc writers write, and the product owner collects requirements. The pressure that an iteration brings requires the team to "blur the lines" a bit. A developer may need to do some testing. A tester may need to write documentation. The team may need to help the PO gather requirements by talking to customers.

What you don't want to happen is for the testers to have more work than they can handle because the developers keep pumping out code. The team needs to *focus and finish*. Customers do not purchase code—they purchase features. They expect that software to be defect-free and reasonably documented.

The Scrum team needs to have the skills necessary to get the job done as well as the authority to do so. There is no development manager role in Scrum. The team has the power to determine the best way to deliver value. They are in charge of their own environment, pace, and even workload.

Before I get into the types of people you may find on a Scrum team, it would be a good idea to take a look at the Scrum values. These are the values of a self-organized team:

- **Commitment**: A Scrum team is not told what to do every iteration. They commit to what they think they can get done, and they commit to satisfying both the DoD and any acceptance criteria. This gives the team a sense of ownership for their work.
- **Respect**: This is a big one. The team needs to respect the Product Owner's input on what work gets done and in what order. The PO must respect the teams enough to trust that they will deliver on their commitments.
- **Courage**: The Scrum Master needs to have the courage to stand up to management and stakeholders while protecting the team's focus. The team needs courage to push back when the PO wants them to overcommit during a Sprint, and the PO needs to have the courage to let stakeholders know the priority of items in the Backlog.
- **Openness**: You will get sick of me talking about transparency by the time you finish this book. Openness can make some folks uncomfortable, but it is truly one of the core values of Agile. Be fully engaged and open—never hide anything from either the business or stakeholders.
- **Focus**: A Scrum team should be laser focused on getting stuff done. Multitask as little as possible and focus on delivering the Sprint goals. The Scrum Master should ensure that all impediments are removed and that the team is shielded from external influences.

So how do the traditional roles fit into a cross-functional team? Let us take a look at each role, and how it changes in Scrum.

The Quality Assurance Engineer

Oh yes, the QA person—potentially the most hated person in the development process. Don't get me wrong; QA folks were a nice enough group back in the Waterfall days.

But their relationship with development was adversarial at best.

It really was a perfect storm of unavoidable failure paired with unrealistic expectations. When a team was working Waterfall there was nothing, *and I*

mean nothing, more important than releasing on time. Success or failure was judged on hitting that date.

And who was usually standing in the way of hitting that all important GA date? The QA engineer.

It wasn't because the QA folks had an axe to grind or that they were holding development to an impossible quality standard. The problem was the way Waterfall dictated the work. Waterfall is a sequential, gated framework. Think of it as a series of events where you can't go to the next event until you finish the one you are currently on.

The QA engineer had the unfortunate responsibility of being the last line of defense when it came to quality of the product, which makes sense—after all, they were all about quality… right?

Not so fast, my friend.

The problem was that in the Waterfall methodology the QA engineers got the code last, after all development was done. They were the last in line. They were looked at as the ones that were standing in the way of developers meeting their all-important date. Another huge issue was that the QA staff was not given enough time to finish all of the testing. It was the nature of Waterfall. You had to hit the date at all costs, and development took longer than expected. Since QA was last, it suffered. The QA engineers were often asked to do what was against their nature. To ignore problems that they found.

This led them to feel like victims. The people who always had to deliver the bad news.

In Agile, the QA engineers are no longer the victims. They are the leaders. They own the quality in an environment where zero defects is the policy, not a target.

Testing should be done as early and as often as possible. Agile teams are required to release with zero defects. This means exactly what it looks like: QA must be taken seriously. What's important is releasing the highest-quality product possible.

QA engineers have the most knowledge about testing on the team, so it is their job to ensure that the QA work is done as early and often as possible. On a cross-functional team, anybody can do the QA work, so the QA engineers need to consult with other members of the team about QA tests, help with the writing of test automation, and assist the PO with acceptance criteria.

The earlier testing starts in the process, the more risk is removed from the process. When I first started working with teams, we were doing what I referred to as "wet Agile." We basically broke the Waterfall work up into Sprints, with lots of really awkward meetings. It was especially nasty for the testers, because the development staff would code and code and code at the beginning of the Sprint,

and then dump an avalanche of work on the testers at the end of the Sprint. It was like Waterfall in a smaller time box. Testing needs to start as early as possible. Test-driven development requires that the tests be written first. You run the test, note how it fails, and then write the code to ensure that the test passes.

The Developers

"When do I get to write code?"

I was working with my first Scrum team, and I was admittedly a hot mess. A developer said this during a standup meeting that was not going well.

I have a little saying I like to repeat to the teams I coach every once in a while. Dairy farmers milk cows, window washers wash windows, and Scrum teams produce value. Notice that I didn't say lines of code. A Scrum team needs to get out of that mode of thinking. Agile dictates that a team does not produce complexity. Agile demands simplicity.

The developer is usually the smartest person in the room. I've had the privilege of working with some amazing developers over the years. People who could look at a problem and create a solution out of thin air. Everybody admires someone like that; however, even a superstar needs to recognize that they can't do it by themselves.

The basketball players with the most offensive skill can't win the game by themselves no matter how hard they try. Think of it—Michael Jordan did not win a championship until Scottie Pippen was traded to the Chicago Bulls. In this brave, new, agile world, it's not about writing great code yourself. It's about making those around you better. Developers need to be about collaboration and mentoring. The truth is that the team is only as smart as the quietest person in the room. Instead of being dominating, development staff should facilitate robust discussions among all team members. This way, the wisdom of the team will surface. Collaboration is not groupthink, it is *healthy conflict.* Everybody on the team should feel comfortable bringing up their point of view and finding commonality on what is being discussed. If not commonality, at least get to where everybody can live with the decision. What you don't want is for team members to just go along with what is happening, but walk out of the meeting expecting failure.

The Scrum team's value to the organization is not in building stuff. The focus is no longer on banging out lines of code; it is on delivering value.

The Scrum Master

We were hosting an event at the office for a group of kids from the Best of the Batch Foundation, which was founded by former Detroit Lion and Pittsburgh Steelers quarterback Charlie Batch. It provides financially challenged youth and their families with opportunities to give their best efforts in all they do

throughout their lives. We were showing the value of Information Technology, and why they should consider a career in IT.

At the end of the day, I displayed some of the talent only a man with hobbies like mine would have. I ripped a license plate in half, rolled up a frying pan, and bent a piece of cold rolled steel in my mouth, as shown in Figure 3-2. I enjoy doing stuff like this for kids, and they in turn were amazed at what the "strong-man geek" was doing. I guess you don't see guys rolling frying pans every day, let alone in a software development lab.

Figure 3-2. Bending a cold rolled steel bar in my mouth. Yes, I have some strange hobbies.

After all of the festivities, one of the children asked me what my role at CA was. When I replied that I was a Scrum Master, the look on her face was price-less. After a couple of seconds of hard thinking, she asked if I was the janitor…

This is not an isolated incident. When I tell people what I do, it's usually met with either puzzled looks, giggles, or a plea for clarification.

I can remember a time when I was having breakfast in a fast-food establishment with my friend Russ Clear. Russ was the person who taught me how to do strongman stunts. He was a former member of the Hell's Angels who became an evangelical strongman and put on demonstrations in churches and schools. Russ was a mountain of a man with a shaved head, tattoos, and muscle on top of muscle. He was not one of those guys who were all show and no go. I personally witnessed him break 10 stacked cement pads with his head. Russ was the guy who taught me how to perform strongman stunts as shown in Figure 3-3.

Figure 3-3. Russ taught me how to ruin a perfectly good frying pan

And I tipped the scales at a svelte 300 pounds at the time. I'm sure we were quite the sight, but we were oblivious to all of the people gawking at us. Finally a guy walked up to us and asked "Are you guys wrestlers?"

Russ shot him a menacing look and said "Nah." He was not a man of many words when he was annoyed, and being mistaken for a professional wrestler annoyed him.

"Well what are you guys, then?" I guess this guy didn't get the hint.

"Well, I'm a Scrum Master," I said.

That ended the conversation.

It's challenging to give a clear picture of what a Scrum Master really does; however, a Scrum Master is vital for the success of a Scrum team. I will do my best to explain the role here.

A Scrum Master is the servant leader of the Scrum team. Notice that I did not say manager. A Scrum Master is not a development manager with a cool, new name. In Scrum, the team is supposed to be self-managed. In other words, they manage themselves. I know, that was a bit redundant, but I need to emphasize that point. The Scrum Master is there to do whatever it takes to make the Scrum team successful—to a point. Being a servant leader doesn't mean that you think less of yourself. It means you think of yourself less.

In my career, I spent a lot of years as a sustaining engineer. In layman's terms, I fixed bugs. I have a natural tendency to want to jump in and fix things. It drives my wife crazy, and I have to fight against myself to not do it with the teams I work with.

Why?

Because if I keep fixing problems for them, they will always look to me to fix them. The Scrum Master needs to help the team to come up with solutions themselves—not do it for them.

To be clear, part of a Scrum Master's job is to remove impediments. It is not to help with coding or design work, or anything like that. Tempting as it may be, you are not there to help the team do their work. You are there to enable them to do their work.

One of the problems with being a servant leader is that it's easy to be a leader. A servant, not so much. Being a servant is a mindset, a conscious decision to put the needs of the team before your own. The team comes first.

And the Scrum Master buys the donuts, as in Figure 3-4. That is just the way it is.

Figure 3-4. These are guaranteed to make folks happy. Some will complain about calories or that donuts are unhealthy; however, they will be gone by the end of the meeting.

The Legs Feed the Wolf

A Scrum Master is the keeper of the Agile flame. A Scrum Master needs to ensure that the team is applying the Agile framework correctly—Scrum in particular. It is vital that a Scrum Master be borderline maniacal about Agile. OK, maybe most Scrum Masters won't go to that degree, but it is important to make sure that the team isn't "doing Agile." They need to be living out the Agile Manifesto every day. This may require that the Scrum Master be able to coach folks on how to apply Agile principles. When I say coach, I don't mean that you scream and yell and throw chairs. I mean to teach, listen, and ask powerful questions, and help the team make decisions without making the decision for them.

One of the things that my wife says all the time is "You don't listen to me."

She's right. In fact most of us really don't listen effectively, even when we are trying really hard.

The problem is that even when I'm listening intently to somebody, I have this internal conversation going on in my head. I'm either trying to solve your problem for you, or thinking of something witty to say, or thinking about lunch

or something. In reality, I think I am listening, but I am doing everything but focusing on what is being said.

Realizing this, the next level of listening is when I'm aware of what is going on inside of my head and recognize when I'm not focusing on the conversation. Stopping myself from "drifting away" enables true communication to occur. The cool thing is that when you get good at this, you begin to see the whole picture. Stuff like body language and facial expressions. This enables me to use everything to understand what the person talking is trying to communicate to me.

A powerful coaching technique is the use of powerful questions. A powerful question gets right to the point. It gives the person being coached the opportunity to clarify, or discover that they know the solution.

Back to my basketball coaching days…

For some unknown reason, my team had a fascination with the right corner of the basketball court. Whenever there was a turnover or fast break (any time we were not running a set play), the girl with the ball would make her way to the right corner of the court. I used to joke that there must have been a puppy there, but this was not a joking matter. While in that corner, the baseline and sideline acted as two additional defenders. This made it easy for the other team to trap the player with the ball. I used to get super frustrated after saying "don't go to the right corner" about a billion times during practice. I should have tried direct questions. It would have looked something like this.

"Jorden," I would say (yes, I'm picking on my daughter), "Do you think taking the ball to the right corner of the court is making it easier to score?"

Knowing my daughter, she would get a very thoughtful look on her face and say, "I don't think so…"

"Why?"

"Because we can't get the ball in the basket."

"Okay," I would say. "What would it look like if we could easily get the ball in the basket?"

"I guess we should get the ball into the paint."

"What do you think is the easiest way to do that?"

"Get the ball to the top of the key?" She asked.

"Yup, that will work better than the corner."

Using powerful questions draws the answer out of people, so instead of telling them, you are helping them figure it out for themselves. This gives the person being coached a sense of ownership that usually leads to them having an easier time implementing the idea.

In my days as a Scrum Master I've found myself having to explain over and over and over again why having a daily standup meeting is a good idea. Finally, I got tired of repeating myself and asked "what would make the standup meeting valuable?"

The person who I was talking to stood there for a couple of minutes with a bewildered look on their face and said "What do you mean?"

I said "If you don't want to attend the daily standup, you must not think there is any value to it. How do we go about making that meeting valuable to you?" That led to a healthy conversation that led to the team making some changes that directly led to improvement in the team dynamic.

One of my sayings is, "Be the Agile Manifesto with skin wrapped around it." Not everybody is going to understand this Agile stuff, let alone embrace it. The Scrum Master needs to be a coach who gently leads everybody down the road towards greater agility. One of my pet peeves is when somebody says we are "doing Agile." In my opinion you don't "do" Agile. A linebacker doesn't show up at practice and say "I'm going to be agile today." He constantly works at it. He needs to develop his agility and constantly work to improve it because as the great hockey coach Herb Brooks used to say, "the legs feed the wolf." If you can't move your feet, you aren't going to play much. You don't flip a switch and instantly become Agile. It's more like turning a dial. Little by little, more and more each day. The Scrum Master is there to lead the Agile transformation and to focus on the successes (what I like to call the beacons) that illuminate the path to agility.

A Scrum Master Should Be the Source of Energy and Motivation

I like to describe myself as "sickeningly positive." A glass-half-full kind of guy. Positivity maps directly to performance; that is a proven fact. What, you don't believe me?

Let's say you are carrying a very full bowl of hot soup across a room. If you keep saying to yourself "I'm going to spill this!" there is a good chance you will. Call it a self-fulfilling prophecy or just dumb luck; your attitude will determine the outcome.

I'm not sure where this story originated from. Truth be told, I first heard it from my pastor, so I'll give him the credit. There were a pair of twin boys. Physically, they were normal and healthy, but their mother worried that they were developing extremely different personalities, as one boy was extremely optimistic about everything and his brother seemed very pessimistic. This behavior got to the point where the boy's mother took them to a psychiatrist to see if anything could be done for them.

After observing them for a while, the psychiatrist attempted to work with the boys. He started with the more pessimistic child. He took him by the hand and showed him a room piled to the ceiling with new toys. He told him that everything in the room was his for the taking. Instead of jumping for joy, the boy dropped to his knees and began to cry.

"What is the matter?" The psychiatrist said. "Don't you like those toys?"

"Oh, sir, they look wonderful and I would love to play with them," said the boy between sobs. "But I'd probably just break them."

The psychiatrist then took the optimistic boy to a room filled to the ceiling with horse manure. Much to everybody's surprise, the boy ran into the room and started frantically digging in the manure with his hands.

"What are you doing?" the psychiatrist yelled…

The boy looked back at him and said "Sir, there must be a pony in here somewhere!"

The Scrum Master needs to be a positive force for change. Keep looking for that pony.

A Scrum Master Is a Facilitator

The Scrum Master facilitates and runs the Agile Scrum ceremonies. Of course, the Scrum Master schedules meetings; however, the idea is not just to hold meetings. Everything possible should be done to make the meetings meaningful and effective. Think about it. Meetings are expensive. Just take a look around the room during your next meeting. All those folks are being paid to be there. Let's try to not waste the time.

In order to achieve success, meetings must be focused and not fall prey to the dreaded "rabbit trail." The biggest complaint that I hear about Scrum, especially in the beginning, is about all of the meetings. I will admit that if the Scrum Master doesn't keep the meetings focused, they can become, well… a train wreck. The idea is to make the meetings valuable. Note that I didn't say that the Scrum Master manages the meeting. They ensure that the meetings happen, and that they are conducted properly.

In order to achieve this goal, the Scrum Master should set up working agreements with the team. I will go into this in further detail later in the book, but here is a good example. As a facilitator, the Scrum Master should encourage the team to define what a consensus is. I like something like this:

"I can live with it, and support it."

You don't have to love it, or really even like it that much. When the team comes to a decision, I expect that when everybody leaves the room they will support it.

We have all taken part in bad meetings. Where you sit in a room and nothing gets done. Where people yell at each other or where one person dominates the room—or worse, where everybody is guarded and afraid to speak out. A facilitator fights against those type of meetings. By using facilitation techniques, a Scrum Master can take part in meetings that have energy. Where everybody is willing to participate and decisions get made. Where everybody agrees that there is value in the meeting.

The real goal is to surface the wisdom of the team. A facilitator creates a safe environment where true face to face communication can happen.

A Scrum Master removes impediments. In the daily standup, a Scrum Master should encourage the team to answer their questions:

- What Did You Do Yesterday ?
- What Do You Plan to Do Today?
- Is Anything Blocking You?

By the way, you really should ask everybody to actually stand up during the standup meeting. Ten percent more oxygen goes to your brain when you are standing up.

A Scrum Master does whatever it takes to make a Scrum team successful. Scrum team members need only worry about generating value. If somebody's computer dies, or they need help with the legal department, the Scrum Master should jump in and get that stuff taken care of.

That doesn't necessarily mean that Scrum Masters take care of everything themselves. They own the problem and if needed get others involved.

The Scrum Master should be pushing for close cooperation across all roles and functions, looking for bottlenecks and lack of transparency. The Scrum Master owns the success of the team process and should be looking for ways to cultivate high performance. Anything that stands in the way of the team being fully functional and productive needs to be addressed and/or removed. For example, a Scrum Master shields the team from external interference. There is always stuff that seems to show up for team members that isn't part of the current Sprint work. A Scrum Master needs to step up and block these interruptions. The team needs to be able to focus on the goals of the current Sprint. The Scrum Master's focus should always be to help the team expedite their progress. To get better every day. By pulling the focus away from the team, the Scrum Master enables the team to work on what is important.

The Scrum Master Is "Team Mom"

Being a Scrum Master can be a thankless job. If the Scrum team is immature when it comes to Agile and Scrum, somebody needs to help them become high-performing Agile maniacs. There is a human side to this as well. To put it bluntly, sometimes team members freak out. Maybe they had a bad night of sleep, or the drive to the office was horrible. Perhaps they went to get coffee and were greeted by an empty pot, or they are just in a bad mood.

A Scrum Master needs to have the people skills to handle all of this. A Scrum Master must know when to jump into a situation, or stay out of it altogether. Sometimes the job requires that you break up fights and or arguments; at other times it requires that you give a word of encouragement. The bottom line is that a Scrum Master is responsible for the happiness of the Scrum team. The Product Owner is responsible for communicating the "Why," and the Scrum Master is responsible for helping the team discover the "How" so that they can build the "What." Expect to have do the impossible to guarantee that everything is working as well as possible.

In this chapter, we explored the three pillars of Scrum, the Scrum roles, and some intra-team dynamics. I also detailed my old powerlifting program and gave a bit of coaching advice. In the next chapter, we will take a look at some Scrum team structures.

CHAPTER

4

Scrum Team Structures

I find that most people don't think of their day-to-day work in the terms of operating within the structure of a team, let alone becoming a high-performing team. Before Agile, the workplace was a very competitive place, where everybody focused on themselves. People worked isolated in cubicles and only came together for status meetings where management looked for whom to chastise because they were behind in their work. Now the focus needs to be on the team and collaboration. Those impacted by decisions make those decisions. Failure is looked at as an opportunity to grow, not to point fingers and shift blame. Information and knowledge is freely shared, and quality is emphasized over meeting a specific date.

That is quite a change in the team dynamic.

Collocated Teams

Members of a Scrum Team need to communicate and collaborate. As I said before, collaboration is healthy conflict. Communication is what makes it so healthy. When you think about it, all of the Scrum ceremonies are designed to foster collaboration. In each one, the team comes together and is encouraged to share different points of view. Collocated teams have fewer barriers to communication and are more productive by nature. If you need to talk to somebody, you walk over to their desk and have a conversation. That is what I refer to as the "million dollar moment." A culture of collaboration is built

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_4

when people can easily come together. I would rather see four or five team members gathered around somebody's desk discussing what they are building than those people head down, banging out code, working in a cubicle farm. A group discussion reaps the benefits of everybody's experience and point of view. Information flows freely in this scenario.

If at all possible, a Scrum Team should be in the same location. Preferably within fifty feet of each other. Why fifty feet? That's something I observed through my own experiences.

I used to work in a building that contained a fitness center. It was convenient to be able to work out during lunch, and I took full advantage of it. I did mostly cardio work. Yes, I do cardio along with my crazy weightlifting and bending iron bars and such. I shudder when I think how fat I would be if I didn't.

There was an outside access door at the front of the fitness center. This door was about fifty feet from the main door of the building. People kept using the fitness center door to enter the building. In the warmer months, this was not a big deal. In winter, it was horrible. I'd be working hard, covered in sweat, and I'd get hit with a cold blast of air every time the door opened. That was annoying, and led to that particular door eventually being locked.

What I find amusing about this incident is that the people using the fitness center door were not saving fifty steps. To get to the elevators, one had to walk through the fitness center and up a hallway. Those same elevators were right behind the main doors. People *perceived* that the fitness center doors were closer; that's why they used them.

If people perceive that it's too much of an effort to get to somebody, they won't go over there. If they sit too far apart, they won't collaborate. This means that a team should not only be in the same office, but also literally sit together. Not in high-walled cubicles or offices, but in an open area that fosters collaborative effort.

The members of the team can easily convey their opinions, convictions, and suggestions with each other. The team can iron out their differences quickly and efficiently because they are sitting together and discussing the problem face to face, which naturally minimizes confusion and misunderstandings. When the team shares a workspace, collaboration becomes second nature. There is nothing more effective than being able to turn around and talk to a person. Posting information about the project on the walls in the team location enhances visibility into the project and makes it easy for everybody to understand the project status. The team needs a common area in order to make posting project information practical. The whole team may not need to take their laptops into a big conference room with flip charts and whiteboards (although I have coached teams that have done this very thing). What is critical is to get the team out of the cubicle farm. Figure out a way to remove the walls between members. When they are no longer isolated, they can easily communicate.

The team is the heart of Scrum. In my opinion, there is nothing more powerful than a motivated Scrum Team. A bunch of individuals working on the same thing is not going to cut it. A true team is collaborative in nature and dedicated to a common goal.

A Scrum Team is not supposed to recognize traditional software development roles (developer, tester, and so on). Don't get hung up on that. Understand that at first the best way for the team to work is in the roles in which they are comfortable. The team can be coached to become more cross-functional as they work toward becoming a high-performing Scrum Team. After all, in the Scrum Team environment, there is no such thing as your work and my work. There is only work for the team to complete, so it's all our work. The team commits to the work each Sprint, and does what it takes to get them done. This creates a certain esprit de corps among the Scrum Team. When I was in the military, that phrase was thrown around quite a bit. Esprit de corps basically means a feeling of loyalty and fellowship among team members. Say what you will, but I know that there is a strong bond between myself and the folks I served with that is still there today. That bond is created when the team works together in the same space day after day.

Distributed Teams

OK, what if it's impossible for the team to be in the same physical location? Sometimes the skill sets required are not in the same location. Some people work from home. Some folks live in different parts of the world.

Obviously, you want the team to be in the same place. How do you create the same sense of shared purpose and commitment when the team is not together?

Use technology as much as possible to get the team communicating face to face as much as possible. Use video conference rooms for meetings and Agile ceremonies. Most laptop computers have integrated webcams. Use them. Do whatever it takes to ensure that the team is communicating effectively. There are tons of collaboration tools to choose from. Don't let that paralyze the team. Pick something, start using it, inspect and adapt. I'm not going to endorse any particular product, because every team is different. Just because something worked for a team I coached does not mean it will work for anybody else. It's up to each individual team to determine what works for them. Technology can't replace bumping into somebody in the coffee room and asking a question, but it can come close.

Guard against the natural "us versus them" tendencies that will arise when the team is not in the same geographic location. If at all possible, I like to bring the team together in the same physical place and do a kickoff meeting or some team building. Even if the team members only meet face to face once, it makes it more comfortable to reach out to one another. The team needs to come together and figure out how to work together.

Emphasize communication and collaboration. To be honest, I've seen people in the same room not communicate. Part of the Scrum Master's job as facilitator is to create an environment where collaboration is possible and gently push the team members to communicate. Look for situations where collaboration would have helped and point them out to the team. It's not easy, but collaboration can happen across an office, city, even the planet.

Part-Time Teams

Think of a sustaining engineering team. They spend most of their time fixing customer defects. This means that they know the code and have the development chops that can help a Scrum project. The problem is that they are event-driven. Most customers don't care that the sustaining engineers are helping out the Scrum Team. They want the bugs fixed.

To me it's a simple exercise in capacity planning. Most sustaining teams that I have worked on were not busy 100 percent of the time. They could devote, say 30 percent of their capacity to Scrum work and still be able to satisfy their sustaining engineering commitments. Transparency is what works best here. Do not overcommit the part-time resources so that a high-priority customer issue wrecks the Sprint. Be realistic about what the part-time team can commit to the Sprint.

Scaling Scrum

To this point, we've been focusing on the Scrum Team. A team of between, say six and ten folks. I hope that by this point in the book you are able to see the value of a small team delivering value in short iterations.

However, what do you do if you need more than ten people?

You don't want to make the team bigger. When a Scrum Team gets too big, communication suffers.

To me, it makes sense to break up the big teams into a bunch of smaller teams.

Feature Teams

It makes sense to organize teams around features. It feels natural. When you think about it, customers buy features. A feature maps to complete user functionality.

A feature team is organized around delivering features instead of focusing on requirements. A feature team delivers complete, working, vertically sliced software each iteration—just like a Scrum Team. The difference is that a feature team works on features end to end, feature after feature. In other words, the team works on the feature until it is complete, delivering vertical slices each iteration. Once that feature is complete, they move on to the next feature. The team has everything it needs to complete the feature. It has the necessary skills, tools, and authority.

Component Teams

While feature teams try to slice stories vertically, component teams are more horizontal across architectural boundaries; for example, database team, user interface team, and so on. I think of feature teams as delivering vertical slices and component teams as providing layers. While it may seem that feature teams are the most attractive, there are good reasons to also have component teams. One of the best reasons I can think of is to build a common service that is used by multiple teams and/or features.

The Scrum-of-Scrums

When you have multiple Scrum teams working on a project, there needs to be alignment and even more transparency. When scaling Scrum, these are achieved with a Scrum-of-Scrums meeting. For the Scrum Team, it's business as usual except that each Scrum Team picks one team member to attend the Scrum-of-Scrums meeting to coordinate the work. The Scrum-of-Scrums meeting is like the daily standup meeting, except that you have members of several different Scrum Teams reporting what their respective teams are doing. Usually the Scrum Team picks the Scrum Master to attend the Scrum-of-Scrums, but anybody from the team can attend. I've even heard of teams rotating who goes to the Scrum-of-Scrums meeting. The goal is to ensure that information is flowing between the teams and back to the team, as seen in Figure 4-1.

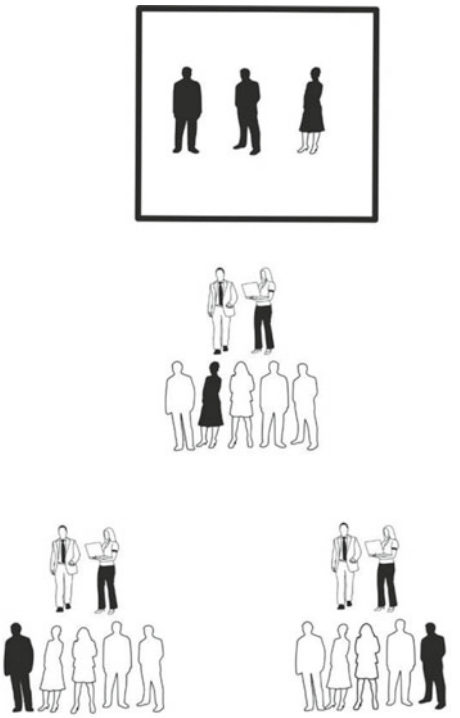

Figure 4-1. The Scrum of Scrums

The idea is that the Scrum-of-Scrums meeting is used to resolve dependencies between teams and determine how teams need to collaborate to ensure a successful release. It can also be used to identify and remove obstacles. The Scrum-of-Scrums meeting does not need to happen daily, but it should have a regular cadence, usually every few days.

The Scrum-of-Scrums meeting resembles a daily standup meeting. Representatives from each team report what their team has completed, what they will work on next, whether what they are doing will impact any other teams, and any team-level impediments. Think of stuff like "I need Team A to finish their story before my team can start this story." These impediments are tracked in a separate backlog. The Scrum-of-Scrums meeting also lasts longer than a daily Scrum meeting. A daily Scrum meeting is not for problem solving. In a Scrum-of-Scrums meeting, the problems are usually between teams and significant. Everyone required to solve the problem is already in the meeting. It makes sense to address it there.

Think of scaling Scrum this way. Everything basically stays the same; it just magnifies on the bigger scale. Instead of a self-guided cross-functional team, you have a bunch of representatives of self-guided cross-functional teams making up a self-guided team of teams. As you can probably imagine, this model can scale to accommodate hundreds of teams.

In this chapter, we looked at the different types of Scrum Teams as well as how to scale Scrum using the Scrum-of-Scrums. Scrum Teams need to be able to communicate and collaborate—even if the members are in geographically separated locations.

In the next chapter, we will explore the Scrum ceremonies and artifacts.

CHAPTER

5

Scrum Ceremonies and Artifacts

It was nineteen ninety-something and I somehow got talked into working for a wild and crazy wrestling outfit. OK, that's a bit of a fib. I love studio wrestling and always have. One of my earliest memories was my grandfather (shown in Figure 5-1) swearing at the television in Italian while watching wrestling.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_5

Figure 5-1. Look at the happy little Scrum Master with his grandparents

No, I didn't wrestle. I will say that I could work the microphone (referred to as "the stick" by those in the business), but I had no interest in being away from my family and working out of a suitcase. A friend of mine owned a gym at the time, and one of the wrestlers lifted weights there. He asked some of us to help out at the wrestling show by working security. He asked that we show up three hours before the show wearing a black blazer.

We showed up at the venue (on time, I might add) and were asked to set up chairs while the road crew set up the ring. The "road crew," surprisingly, were wrestlers. Let me tell you right now that all of those rumors about professional wrestling rings having extra padding were false.

Once the show started, our job was to make sure that nobody threw anything into the ring, and to control the crowd when the action spilled out of the squared circle.

Simple enough, except that the action went into the crowd during every match…

The whole show I kept my eye on two adorable little old ladies sitting to the right of the ring. I had no idea what they were doing there. My grandfather used to laugh at "ringside Rosie," a little old lady who would sit ringside and

yell at the wrestlers. This was different. I was worried that they would get trampled if the action got too close. Luckily nothing got too close to them—until the main event.

The main event was a brawl from the get-go. At one point, the action spilled into the crowd, right in front of the two ladies. I did my best to keep the crowd back, with my back to them. Then I heard, "Here honey, take this!" I turned around to see both ladies offering the wrestlers cookie sheets, which they gladly took and began to use as weapons.

I'm not sure why I shared that story. I guess it shows that even with the best of intentions, sometimes things don't turn out as you expect that they might.

That's why the Scrum artifacts are important to take a look at regularly.

If you are using the Scrum framework, you should be producing stuff. Yes, you are churning out value for your stakeholders, but you also generate by-products that reflect the health of the Scrum team. These are called the Scrum *artifacts*.

The Definition of "Done"

How does a Scrum team know when they are done? I guess it would make sense to define what "done" means, because everybody may or may not have the same opinion of what that word truly means. This reminds me of a famous quote:

> *It ain't over 'till it's over.*
>
> —Yogi Berra

The Definition of Done (DoD) allows the team as a whole to agree on what it means to be… done. Aligning everyone's understanding of what "done" truly means affects the work at the story level (for example, "done" means coded, tested, and documented). Everybody's expectations are met, and there are no surprises, gaps, or misunderstandings as the team works through the Sprint. The DoD is a list of technical and professional completeness. One thing to keep in mind is that the DoD needs to be separate from the story acceptance criteria. Acceptance criteria are specific to each story and will vary from story to story. They should focus on the expected or correct outcome of the story. The DoD is applied to all stories that the team works on. It does not change; however, portions of the DoD may not be applicable to all stories. Some teams list "acceptance criteria met" as the first item in the DoD.

You might find that confusing. I'll explain further, but before I do, this is a good time to talk about how a team comes to an agreement.

Consensus

I touched on this before. Most folks have a skewed notion of what consensus means. To most, "my way of the highway" covers it. This leads to arguments, people digging their heels in, or worse—apathy. How many times have you seen people just sit there, maybe rolling their eyes at the events unfolding in front of them. They may go along with whatever is decided, but they will walk out of the meeting muttering "no way" under their breath.

Not what I would call healthy team dynamics.

One of the things that I picked up hanging around the wrestlers as seen in Figure 5-2 was their lingo. They spoke their own language when they were around each other, mostly because they didn't want the fans (which they called "marks") to catch on to what they were talking about.

Figure 5-2. My alter-ego "Captain Agile"

A *shoot* was defined as "brutal reality" or the truth. Legend has it that the term "shoot" came from Japan. When two wrestlers squared off and actually fought (for real… as in trying to hurt each other), it was called shoot wrestling. Wrestlers would say things like "That was a straight-up shoot" or "I'm shooting with you right now" if they were being truthful.

A *work* was defined as make-believe. A lie. If you were being worked, it meant that somebody was being less than truthful in an attempt to get over on you.

When it comes to team meetings, in wrestling terms, I want everybody in a shoot. Making the meeting a work doesn't help anybody. The team needs to be comfortable and honest in order to come to a consensus. Open and honest conversation unearths problems and leads to better results.

I like to define consensus as "I'm not going to torpedo this decision." Seriously, things may not turn out as I expect. I may not get my way, but if I can live with the decision, I will support it. Once a team reaches consensus, everybody needs to straighten themselves, clean the blood off othe walls, and speak with one voice… *the voice of the team.*

OK, I was joking about the blood on the walls thing (kind of). Meetings can get ugly. One of the techniques that can help a team reach consensus is called the "fist of five." Yes, it sounds like a really bad martial arts movie.

To facilitate the fist of five, ask the team to vote on an idea by holding up fingers (Figure 5-3).

Figure 5-3. Five fingers

Five fingers stands for total support. By total support I mean "this is the best idea EVER" kind of support. You don't just like it—you want to marry it and have babies. Obviously, you should rarely see any fives. If you do, you may have to explain the process again. Or your team is a bunch of overly dramatic smart alecks…

Four means that you like the idea (Figure 5-4). You feel that it is something you can get behind. I've heard it explained that you wish you had thought of it. I think that explains the four the best.

Figure 5-4. Four fingers

Three is the definition of consensus (Figure 5-5). You can live with it, and you will support the decision. That means that you do not walk out of the meeting muttering "no friggin way" under your breath.

Figure 5-5. Three fingers

Two means that you have some issues with the idea and would like to discuss it further (Figure 5-6). You aren't totally against it, but not totally for it either.

Figure 5-6. Two fingers

One finger means "over my dead body" (Figure 5-7). You have major issues with the idea and will not move forward.

Figure 5-7. One finger

One should always be signaled with the index finger, no matter how upset you may be… if you get my drift.

As a facilitator, you are looking for ones and twos. They are an opportunity to delve deeper into what is being discussed and further explore whether an agreement can be reached. It isn't so much that folks are being inflexible. It's about finding consensus so that the team can move forward.

Back to the Definition of "Done" (DoD). This is possibly the most important thing to a Scrum Team. Remember, the team is to take ownership of themselves. They are responsible for their own accountability. If they are to be truly self-directed, the team must come to consensus on a DoD, and then work to it. In other words, the team strives to satisfy the DoD and acceptance criteria of each individual story. There is nobody looking over the team's back anymore to ensure that everything is getting done. The team defines what done is and works to that definition.

It is tempting for a Product Owner to put things in the acceptance criteria that really belong in the DoD. For example, I remember a situation where a Product Owner I was working with put the phrase "all QA tests written, executed, and passed" in the acceptance criteria. When I asked why it was there, the Product Owner said that QA testing was vital, and the focus needed to be there.

If the team has agreed that the QA tests (both manual and automated) must be written and passed as part of the DoD, then the team and PO can both be sure that when a story has been marked as done, the QA has been done.

There is no need to revisit anything. If the story is done, everything that applied in the DoD is done. The Product Owner needs to trust that the team will satisfy all applicable items in the DoD for each story, and the team needs to work to the DoD in good faith.

The DoD needs to be posted in a public place and take into account all functional and nonfunctional criteria.

Functional requirements are the things you would expect. Specific functionality that the team is building. For example, a football helmet should be able to withstand multiple strong impacts. Nonfunctional requirements are more about the system. Stuff like scalability and performance. A nonfunctional requirement for a football helmet would be that it comes in different sizes so that it fits a variety of football players.

Why post the DoD in a public place, as in Figure 5-8?

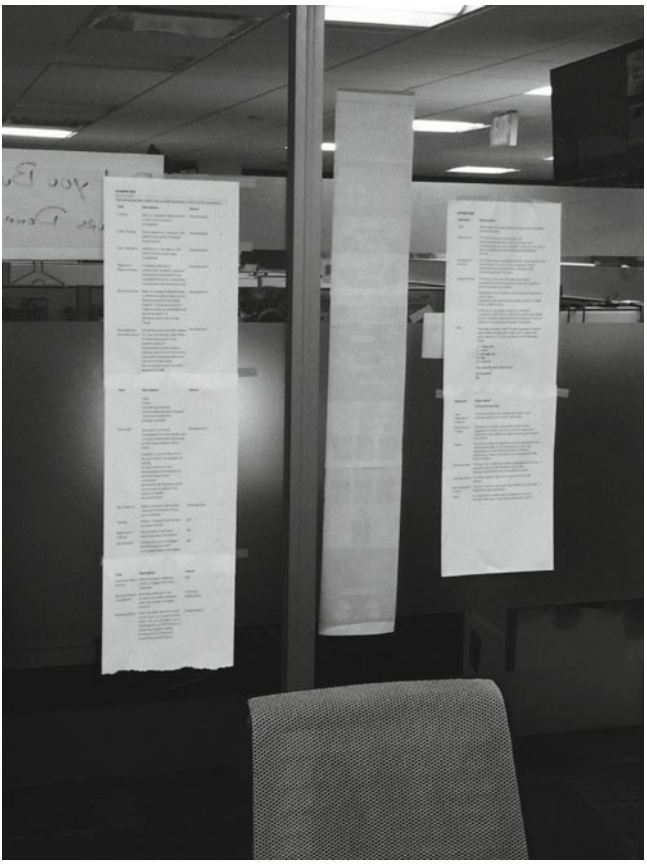

Figure 5-8. A Definition of Done and Definition of Ready taped to the wall where one of my teams holds the daily standup meeting

This way, the team is constantly reminded of it. It might seem trivial, or even something you would do in kindergarten. However, we are radically changing the way the team works. Remember, nobody is looking after them to make sure everything gets done. It is now up to the team, and they use the DoD to ensure that nothing is forgotten.

Let's take a look at a sample DoD, shown in Table 5-1.

Table 5-1. A sample DoD (Definition of Done)

	Criteria	Check-off
1	Design complete	Design has been reviewed.
2	Code complete	Code has passed a team code review.
3	Development testing complete	100% code coverage with tests; all tests pass.
4	Documentation completed	Doc writer passed tests.
5	QA turnover process complete	Code has been promoted in source management. QA regression test has been updated.
6	QA testing complete	Zero defects.
7	Regression testing complete	All regression tests have been run and passed.

This is a simple Definition of Done. Everything here could possibly apply to a story. If something does not, it simply is not applicable. This is how the DoD can apply to all stories. Every item in the DoD may not apply uniformly to each story. For example, not every story may require documentation. The idea is that the team discusses the DoD and determines that documentation doesn't apply here. Having this type of conversation is crucial for the team to deliver value effectively, Sprint after Sprint.

Individuals and Interactions over Processes and Tools

One person may think that the functionality in the story needs to be documented, but others on the team may point out that the functionality doesn't need to be surfaced to the end user, or already has been put in the documentation. Coming to consensus as a team as to what "done" looks like, and executing that, ensures that everybody's expectations are in line with what will be produced. After all, building software is extremely complex. This requires the team to be adaptive in nature. Consensus-building is an example of how Scrum recognizes the need for adaptability, and encourages the team to solve problems collaboratively as they build value. Because of this, all of the elements in the DoD should be discussed and checked before the team declares anything done. The DoD makes it easier for the team to build the right thing—correctly.

The Definition of Done also needs to be a living document that grows and changes as the team evolves. Table 5-2 is an example of a more complex DoD. All of the tasks need to be considered before a story can be completed.

Table 5-2. A More Complex DoD

Task	Description	Owner
Coding	All code (new or modified) pertaining to the story has been written.	Development
Code Testing	Story owner has reviewed and tested new and/or changed functionality.	Development
User Interface	User interface work has been completed.	Development
Demonstration	Story functionality can be demonstrated to the stakeholders.	Team
Development Documentation	All necessary documentation has been added.	Development
Delivered	QA and/or Technical Publications has been given one or more of the following: A build or a set of libraries that is ready for testing An environment in which testing can be completed Refreshed help libraries Current documentation	Development
Zero Defects	All defects have been resolved.	Development/QA
Testing	New or changed functionality has been tested.	QA
Regression Testing	Regression tests have been completed.	QA
Automation	Tests have been automated where applicable.	QA
Documentation Review	Documentation additions and/or changes have been reviewed.	QA
Documentation Completion	Customer-facing documentation has been completed.	Technical Publications
Team Demonstration	A demo has been done for the team.	Team

OK, so the team has come to a consensus on a definition of "done." It is posted in a visible place. How does it get applied? It can be helpful to add checkboxes to the DoD. This makes the document a checklist. As the team satisfies DoD items, they are checked in the box.

That may work for a more mature team, but I like to overemphasize the DoD—especially for teams that are new to the concept. I like to have a quick "DoD rally" when the team declares a story done in the stand-up. When this happens, I walk the team through each item in the DoD and use the fist of five to get consensus. As usual, I'm looking for somebody to question (or at least

waver on) one of the items I'm shouting out. This leads to a conversation and deeper understanding of the story status for all involved.

An extra added bonus here is that the DoD makes release planning easier. A team working to satisfy the DoD is finishing all of the stuff that usually gets pushed aside to get through the gates in Waterfall (testing, anyone?) is now done every Sprint. This enables more reliable forecasting so that a better product is released.

And that is straight-up shooting…

The Definition of Ready

Before take-off, every pilot goes through a pre-flight checklist to ensure that the aircraft is capable of making the upcoming flight. It's also a good way to ensure that nobody has forgotten anything. You don't want to get up in the air and realize there is a problem with the aircraft that could have been avoided.

It's the same thing with the Definition of Ready (DoR). How do you know when a user story is ready to be pulled into a Sprint? Just as the DoD helps to define when a story is done, the DoR helps to define when a story is ready to be pulled into a Sprint.

The story needs to be properly sized. Most Scrum teams size their user stories with story points. Yes, story points. I can almost hear you rolling your eyes right now. Most people find story points confusing, but that's because they are thinking about them the wrong way.

My wife hates it when we need to go somewhere, and I'm in the middle of something. In this scenario I'm usually into whatever I'm doing up to my elbows, and running late. The conversation usually goes like this:

"How much longer do you think this will take?" my wife asks.

"Ten minutes or so…" I'll reply....

Half an hour later I'm still not ready, and my wife is not amused.

The issue here is not that I'm lying. The problem is that we human beings are terrible at estimating how long it will take to do something. Think of it this way—how big a job would it be for you to paint your living room?

Note that I didn't ask you how long it would take, or what you had to do to get the job done. All I want to know is how big the job is.

So let's say that we decide painting the living room is a large job. Knowing this, how big a job is painting the master bedroom?

Welcome to Agile sizing.

So why do we do sizing and planning the way we do in Agile? I mean, doesn't it make more sense to meticulously plan everything out like we used to do in Waterfall?

Not really.

The reason we use what may be considered an inexact process to determine how much work can be done by a team is the way stories and/or tasks relate to each other. In something as complex as a development cycle, it's a safe bet that stories and tasks do not stand alone. They relate to each other. In other words, you normally just don't go to work on story 3. You have to finish story 1 and part of story 2 before you can start story 3. Knowing this, when one story takes longer than expected to complete, it is not reasonable to think that another related story in the Sprint will take less time—the reality is that normally all remaining tasks will take longer.

As long as you are consistent in the way you size your stories, everything will work out fine.

More on story pointing later. What I wanted to point out here is that stories should be sized before they show up in a Sprint. If not, the team will need to get together and size the story before it can be worked on, wasting time that should have been used to produce value.

Back to the Definition of Ready (DoR). By ready, we mean *ready* ready, not kind of ready, or sort of ready.

The DoR is a collection of everything necessary for a user story to be developed. Just like the DoD, the DoR is likely to change and evolve through the release, but there should be only one that is used for all user stories. If the DoR is properly defined, everybody on the team should be comfortable with it when it is pulled into any Sprint. As a rule of thumb, a user story should follow the INVEST scale:

I: Independent

N: Negotiable

V: Valuable

E: Estimable

S: Small

T: Testable

Using INVEST, the team can ensure that the story is well defined. Don't stop there when building the DoR; think about what your particular team situation requires.

A simple DoR might look like Table 5-3.

Table 5-3. A Sample DoR

Element	Description
Title	Meaningful, short, and to the point. A customer or stakeholder should be able to understand what the story is about from the title.
Description	Everyone on the Scrum team can understand and be able to explain what the story is about. If at all possible, it should be written in the canonical format.
Acceptance Criteria	The Product Owner's specific requirements that must be met for a story to be completed. Those requirements must be detailed in the story and communicated to the team.
Dependencies	The story description lists any dependencies and/or prerequisites that are required to start or complete a story. Example: Installation of any software and or services required to start a story, or any training required.
Size	The story has been sized. The work required is the relative size of effort for the entire team, not in actual time units. Select {1, 3, 5, 10, or 20} from the following scale: 1 Teeny tiny 3 Small 5 Average size 10 Big 20 Massive (and probably not able to fit in a Sprint) The collective work required by: Development QA Documentation
Owner	The story has been championed by someone on the Scrum team. This person ensures that the story gets done.
Demonstrable	The story has a method in which completed work can be demonstrated to stakeholders at the Sprint Review.
Testing Criteria	The story contains requirements for QA testing.
Documentation Criteria	The story contains requirements for documentation.

A good DoR will ensure that everybody is clear about what is expected, the team has what it needs to be successful, and nothing has been forgotten before it gets pulled into the Sprint.

The Backlog

The Backlog is where all the good stuff lives. The Backlog is a repository of all functional and nonfunctional requirements (plus other stuff) reflected in user stories. Those requirements show up as features, both new and as changes or fixes to the existing product. The Backlog is owned by the Product Owner. That means the PO has the responsibility of maintaining and prioritizing the user stories in the Backlog.

Note that I didn't say that the Product Owner *writes* all of the user stories in the Backlog. Anybody can write a user story; it's up to the PO to know what is going into and coming out of the Backlog.

Let's take a closer look at the user story. A user story is unit of work that creates value from the customer's viewpoint. A user story usually contains a simple canonical statement that looks like this:

As a *user,* I want *something*, so I can *benefit.*

The *user* is the person the story is intended for. I suggest that you don't use the word *user* here (*as a user, I want*…). You want the Scrum team to look at the story from the perspective of the person who is going to be working with the software. The term I like to drive home is *the person on the glass*. In other words, the guy in the data center who uses this software in ways we never thought of to solve problems that we never heard of.

There is that phrase again. You will read it a few more times in this book because I feel it's that important. I was involved with a project where for 6 months I interacted with IT management to build a solution for their data center. Every time we showed them what we were building, they loved it… well at least until we showed the operators in the data center. Their reaction was

"We won't use that…"

And that was it. Management may have loved it, but the guys who had to use it put it on the shelf. Months of work down the tubes because we were listening to the wrong folks.

Doesn't it make sense to get feedback from the person who actually uses the product?

To me it's a no-brainer, but are we doing it? So many times in my experience I see management, or the VP of purchasing, or somebody waaaay up the chain of command interacting with a Scrum Team or Product Owner.

Think of it this way: If I opened up a lemonade stand, would you ask the kids who are drinking the lemonade what it tastes like, or would you ask the parents who bought the lemonade for their kids?

I'm sure that the parents would say stuff like:

"I don't like them having that much sugar."

"You are charging too much."

"I don't like the way your booth looks."

Notice, nothing about what the drink you are selling tastes like. In reality, there is no difference with software. We are trying to make the people who use our software our fans—not the folks who manage those who use our software.

But shouldn't the team focus on those who control the purse strings? You know, the guy who makes the purchasing agreement?

Let me answer that question with a question. Do you think that the folks in the data center (on the glass) are happy when you take away the software that they like to use? From what I've seen at customer sites, when the users aren't happy, nobody is happy.

If we are getting feedback from anybody but the users "on the glass," is it really valuable?

A user story forces the team to focus not just on what functionality they are trying to deliver or what problem they are trying to solve. They must also think about who wants the functionality or is trying to solve the problem. Knowing this, the team and Product Owner should have a conversation about who the user is and how that user performs the day–to-day work required for his job. This is why *personas* are used. We will talk about personas in detail later in the book. For right now, let's just say that we create imaginary users and plug them into the stories to help the team envision how real, flesh–and-blood people in the field will use the software being built by the team.

To get back to the user story's canonical statement, the *something* is a need or feature. It is the functionality that is desired by the user. This is what the team will build and deliver.

The *benefit* is why the story was written in the first place. It is what the user will use the feature to accomplish. It is the value that will be realized by building this thing.

And that's important.

I've ruined a lot of meetings by asking a simple question while the team and Product Owner were going on and on about the features that were going to be built by saying one little sentence:

"Why would a user want this?"

If you can't explain why the customer wants what is in the user story, it probably isn't needed no matter how cool the Product Owner or team thinks it is.

A user story needs to be able to stand on its own. When developers first start to organize work into stories, they are tempted to split the work into chunks that make sense to somebody who builds software. For example, a story for the back-end work, a story for the database work, and a story for the user interface. In reality, a story needs all of these things to stand alone, so maybe you do just enough of the back-end work to make use of a mocked-up database that drives sample data through the user interface so that the stakeholders can see what the product will look like. Next Sprint, maybe the team can put a real database in so the user can see real data.

This is referred to as vertical slicing. Vertical slicing allows a story to be a demonstrable unit of work that a customer can appreciate and understand. More on that when we talk more in depth about writing stories.

A user story must also be sized so that the work can be completed in a Sprint. That means that it is testable, documented, and can be shown in a demo. It must satisfy both the Definition of Done and acceptance criteria.

The Backlog can also contain defects and technical debt.

Bugs are the inevitable truth of software development. No matter how hard you try, things will go wrong—and you will have bugs to fix. In the Scrum world these are called *defects*.

As you have seen, there are quite a few meetings and rituals in Scrum. What most people forget is that Waterfall had its rituals as well. One of my favorites was when development and support would sit around a table and argue about the severity of bugs. Back in those days, you couldn't release a product unless all severity 1 and 2 QA issues were closed. Obviously, development wanted to make their release date, and QA wanted to ensure that the product was of high quality. What resulted was a negotiation session that would rival the most cantankerous contract negotiation you can think of…

In Agile it's much easier—the standard is "zero defects."

Which means zero defects. As in no new bugs: Zero, none, zip, nada, nichts.

The idea is that a defect does not increase in value over time, but it does increase in cost, *exponentially*. The standard that I have seen is that a defect

should be closed in the Sprint in which it was found. The originating story does not close until the defect is fixed. The idea is that we do not, under any circumstances, want to build a mountain of bugs.

OK, that's fine and all, but what do we do with all of this stuff that has been in our products for years? Like the stuff that was "negotiated down" back in the Waterfall days that we never got around to cleaning up?

That, my friends, is *technical debt*.

It's thought of exactly like it sounds. If you don't have the money to buy something you want right now, you go into debt to get it. The funny thing is that once the novelty of that shiny new thing wears off—you still need to pay for it. The longer it takes to pay that debt off, the more you pay for it in interest.

It's the same thing with technical debt; it's much more expensive to correct a bug once a product has been released. You need to rip the modules back open, re-learn how the code behaves, and if a customer finds it, move heaven and Earth to get it fixed quickly and keep them happy.

Every product has some measure of technical debt. Just like defects, technical debt needs to be identified and attacked. You certainly can't give it as high a priority as a defect, but Product Owners should be encouraging their Scrum teams to be taking on a bit of technical debt every Sprint. The number that I personally like is 30 percent of team capacity.

I'd be remiss if I didn't also say that stuff like writing automated tests is also technical debt. In my mind, technical debt is really anything that does not provide value to the customer. Technically (sorry for the pun), defects are debt, too—we just try to "pay them off" immediately.

This is why both defects and technical debt should not be assigned story points. They should negatively impact velocity and give an accurate picture of how fast the team is getting features done.

What is velocity, you ask? I like how my friend Bob Carpenter defines it. Velocity is how fast a Scrum Team delivers value to customers.

Let's say that you decide that you want to start marking things off of your bucket list. For those of you who don't know, a "bucket list" is a list of things that you want to do before you… ahem, kick the bucket.

If running a marathon is on your bucket list, as shown in Figure 5-9, you may want to answer some questions. For example, how long will it take you to run the race?

Figure 5-9. Jim Kokoszynski at the Pittsburgh marathon. Yes, he finished. That means he ran approximately 25.2 miles more than I would have

To figure this out, you need to know how fast you can run a mile…

That is not going to work. A marathon is 26.2 miles long. You need to estimate what your pace per mile will be over this distance. Normally, you don't do a run longer than 20 miles during a marathon training cycle. So, how do you guesstimate your marathon pace? Do you pull that number out of the air?

What most folks do use is a pace calculator. You plug in your time from a recent race (say, a 5k), and it will estimate how long it will take you to finish the longer races.

This is how velocity works in Agile.

Velocity is defined as how long it takes teams to get Backlog items done. In other words, it's how quickly the Scrum team can deliver value to customers. I've already talked about story sizing. An Agile team doesn't go through the story sizing exercise for the experience; it is done so that we can determine when the project will be ready for release.

It works like this. As a team you have a Backlog of stories and epics (which are in reality really big stories or ideas that need to be broken down). During release planning, items get moved in to a more focused Release Backlog. It's not really necessary for all of this stuff to be fully groomed—in fact the goal should be that only two or three Sprints worth of stories are fully groomed.

This may not make sense to you yet, but it's a good idea. Remember, we want to be fluid and react to feedback. Why spend time grooming stuff you may or may not be changing (or even doing)?

The key is that everything in the release Backlog is estimated. All stories and epics have point values. This gives you a number. For simplicity, let's say your release Backlog contains 500 story points. As your team works through Sprints, you will get a pretty good picture of how many story points they will be able to complete per iteration; that's your team velocity.

So if your team's velocity is 50, it should take you 10 Sprints to complete what's in the release Backlog. Average velocity will stay constant over the duration of a project—more or less. Velocity might go up over time as the team makes improvements. When a team is new to Scrum, velocity may fluctuate wildly at first, but the average will settle out in a few Sprints. This makes velocity useful for predicting when a project will end. If the work in the Release Backlog changes (for example the Product Owner adds stories and increases the number of story points based on customer feedback), velocity will tell you when the team will be done. This is why velocity is defined in units of value (stories) as opposed to units of effort (tasks). You get an accurate picture of when the team can realistically complete the work.

Gone are the days when the development manager would set the release date. In Agile, the ability of the team to deliver value is what will make that determination.

Back to defects for a moment. A Scrum team needs to take the zero-defects policy very seriously. As I've said before, a Scrum team needs to be as transparent as possible about everything, including defects. When a defect is identified, it needs to be tracked with the rest of the work in the Sprint. It should go into the Sprint Backlog and be given the highest priority possible. That is a judgment call, but I would go as far as to say that if the current defects found in the Sprint are serious enough, Sprint work should stop if the team feels that it can't effectively address them.

Now you that you know what is in the Backlog, what does a good one look like? A healthy Backlog is refined and prioritized. As stories move up in priority, they become better defined (or refined).

Product Owners should guard against their Backlogs becoming what I refer to as "obese Backlogs." An obese Backlog is one that is big, bloated, and unwieldy. It starts out with the best of intentions. All of these stakeholder requirements are in there because we want to implement them… someday. Think of it like my closet. My weight tends to… fluctuate a bit. My closet contains "fat" clothing that I wear currently. I also have clothing in there that is too small for me: just in case I decide to go on a diet and lose bunch of weight. So my closet is a jammed-up mess of clothes—most of which I will never wear as seen in Figure 5-10.

Figure 5-10. Yes, this is my closet. Full of stuff I rarely wear

Sure, I keep these clothes around because I hate to get rid of them—after all, I spent good money for them. Truth be told, even if I do get to the point where I fit in those clothes, they will be out of style.

So why are they taking up space in my closet?

It's the same thing with the Backlog. Why do we have all of these things that we may never do cluttering things up?

I can remember having a conversation with a Product Owner about the size of his Backlog. At the time, his Backlog had over 400 stories in it. Definitely obese Backlog territory. I talked to him about getting rid of some of the stuff in his Backlog, and it did not go well.

"That's all stuff that a customer took the time to tell us they wanted," he said.

"OK," I shot back, "but you aren't going to have the time to do all that stuff in four releases, let alone the one we are currently working for on."

"But customers asked for it…"

In a perfect world, we could deliver every little thing a customer asked for. I'd also ride a unicorn to work, but that's neither here nor there. The reality is that there is a finite amount of stuff a Scrum team can produce. The Backlog needs to reflect this fact so that the team can have a sense of what the priority is, what may be coming in the future, and unanticipated work. A big, fat Backlog makes it very difficult to do this. I'm not opposed to the Product Owner creating a "just in case" Backlog that contains very-low-priority stories that might gain traction someday. That being said, I still question keeping that stuff around. Remember, you have at least a Release Backlog, which contains what you plan to work on in this release, and a Sprint Backlog, which contains stuff you plan to complete in the current Sprint. Most teams also have a general Backlog that contains stuff that is important, but not in the release being currently worked on.

Burn-Down and Burn-Up charts

Burn-down and burn-up charts are used to show the progress of a Sprint or release. A burn-down chart shows the effort remaining in a Sprint. A burn-up chart shows what has been delivered to this point in time in the project.

Let's go back and talk about stories again. When a story is pulled into a Sprint, it should satisfy the Definition of Ready. For our purposes, we will assume that the story is appropriately sized. During the Sprint Planning meeting, the team looks at the story and breaks it into tasks. Tasks are segments of work and are figured in hours.

But not as you would think. Agile teams work in "ideal days."

I've lived most of my life in a Western Pennsylvania mill town named Aliquippa. The steel mills are long gone, but the work ethic is still here. An honest day's work for an honest day's pay is a mantra I've heard my whole life.

An ideal day for a software worker is about five hours. Believe me, people have a hard time with that one. I know that I've been conditioned to believe that if you put in less than eight or nine hours a day, you were cheating. An ideal day is not where you put in five hours and go home. To be clear, I'm not suggesting that you spend five hours a day working and the rest updating your Facebook profile. What we are trying to do is be "honest" about our honest day of work. Truth be told, we never sit down and spend eight hours a day banging out code or testing. We go to meetings, deal with our email, talk to our teammates about design or trouble we are having with what we are working on, attend town hall meetings, get coffee, visit the restroom, and a bunch of other stuff I won't detail here. The five hours should reflect how much time we spend actually working on user stories a day.

Take your ideal days and multiply them by how many working days there are in the upcoming Sprint. Most two-week Sprints will have ten working days. Now subtract one for all of the Agile ceremonies (which we haven't talked about yet), and you end up with around 45 hours of available capacity for the Sprint. Obviously, if you are planning on taking any days off during the Sprint, it will decrease your capacity.

Add up everybody's capacity and now you have an idea of how much work the team can commit to during the Sprint. During the Sprint Planning meeting (or ceremony) the team, PO, and Scrum Master will take the stories that are pulled into the Sprint and decompose them into tasks. I'll get deeper into how this is done later in this chapter, but for now just understand that a story will be broken down into multiple tasks, and these tasks will be assigned to team members and estimated. Task estimation is done in hours. Going back to the "Paint the Living Room" example, I might have a task titled "Paint the ceiling" that I estimate would take two hours. Another task might be "Put drop cloths over everything," which someone else takes and estimates at one hour.

All of these hours will add up to reflect how much work is required in the Sprint. The team is careful not to go over the team capacity, and team members are careful not to commit to more than their personal capacity will allow. As work is done, these hours are "burned down" or taken off the number that reflects how much work is left to do in the Sprint. The Scrum Master takes this information and publishes a burn-down chart each day, as seen in Figure 5-11. The burn-down chart shows how much work is remaining in the current iteration.

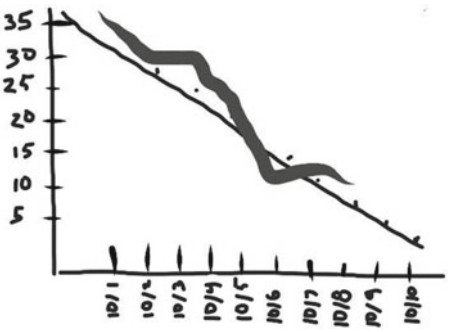

Figure 5-11. A sample Sprint burn-down chart

The burn-down chart should be shown to the team daily. It gives an accurate indication of how much work is done, and it does a good job of predicting whether the team will finish all of the work it committed to for the current iteration.

The release burn-up works on the same principle, except with story points. As the team completes stories, the story points associated with those stories are "burned up." After three or so Sprints, the team should have a reliable velocity. As we talked about before, velocity is the average number of story points the team can complete in a Sprint. The Scrum Master publishes a release burn-up chart that shows what story points the team has burned up, as well as how many story points are left in the release Backlog.

Daily Stand-up or Daily Scrum Meeting

The daily Scrum, or stand-up meeting, is probably the one Agile ceremony that causes the greatest amount of angst. It seems that people just can't wrap their minds around the concept of a short, daily meeting. In fact, they freak out about it.

I can understand why this happens. In my opinion, one of the pillars of command–and-control culture is the status meeting. In a Waterfall status meeting, the development manager would require every person on the team to give a status update. You know, what they have been up to since the last status meeting. It seems to make sense; however, that's not what happened. The problem with command–and-control is that it gives you exactly what you expect, but not necessarily what you want. Since the person who directly controls your review (and your paycheck) is running the meeting, the people in the meeting try to tell the development manager what he wants to hear. In other words, they try to cover their collective asses.

Here is another piece of professional wrestling lingo for you. The word *sell* means to act. To make the audience believe what is happening in the ring is something that it is not. For example, my opponent swings wildly and lands what amounts to a love tap on my jaw, but I act as if he just knocked four of my teeth out. If you watch a professional wrestling match closely, you will notice that almost everything is exaggerated. From facial expressions to the way holds are applied to how wrestlers move around the ring and gesture to the crowd—it's all about selling what you are doing to the audience. If a wrestler didn't act like his opponent's moves affected him much, he would be accused of not selling. Worse yet would be if you were accused of overselling. This is where you acted up a bit too much and made things look unrealistic.

Everybody in a status meeting is selling to their boss, and this creates some problems. Most status meetings suffer from an overall lack of transparency. Nobody wants to give the development manager bad news, so they hide it for as long as possible. They hope to fix it by the next status meeting. The development manager and tech lead make decisions about what needs to be done at this meeting. By nature, most folks don't like being told what to do, so this usually doesn't sit well with the team. Status meetings are also long,

arduous affairs because everybody needs to sit around and listen to each update from everyone on the team. "Listen" might be too strong a word. Truth be told, most people in a status meeting aren't really listening. They are either thinking about what they will say when it is their turn or pretending to care about what is going on.

The daily stand-up is not a status meeting. It is a commitment meeting. Instead of the team telling a manager what they are doing, they make commitments to each other. Every 24 hours, the team gets together and they answer three questions:

- What did I do yesterday?
- What do I plan to do today?
- What is blocking me?

Answering these three questions is an awesome way for the team to understand exactly what Sprint work has been done and what remains. When a team member says what she plans to do today, she is making a commitment to her team. If others on the team can help out, they offer their services—making commitments. When people commit to a task, there is a sense of pride and ownership. This is a totally different dynamic from a manager divvying out work. Progress is measured daily as team members detail what was accomplished the previous day.

Anything blocking a team member's progress is addressed by the Scrum Master. This way, the team member doesn't have to worry about handling the issue. It is now the Scrum Master's responsibility. The team member can now focus on Sprint work. For example, if a team member is having issues with their laptop computer, it is brought up as an impediment at the daily stand-up, and the Scrum Master takes care of getting the laptop repaired.

The daily stand-up meeting should take no more than fifteen minutes. It is not a place for problem solving, planning, or design. Also, only team members may speak. To be clear, anybody can attend the daily stand-up. It is an excellent way to see where things stand with the project, but the meeting belongs to the team.

If something comes up that goes beyond the spirit and intent of the daily stand-up meeting, it needs to be moved to the parking lot. The *parking lot* is where the team "parks" subjects to be discussed later. I usually hang a big piece of paper on the wall where the team holds the daily stand-up meeting and put sticky notes on it when a parking lot item arises.

And the parking lot is always empty when everybody goes home. When a sticky note goes on the parking lot paper, I expect it to be taken care of that day.

Backlog Refinement

Backlog *refinement* (or *grooming*, as it was previously known) is where the team gets together with the Product Owner and Scrum Master and work on stories in the Backlog. The Backlog needs to be visible, organized, and current. Requirements change quickly, and the Backlog (or Backlogs) must reflect that. The Product Owner needs to be constantly evaluating the Backlog. As requirements change, stories may need to change in priority. As stories are prioritized higher and move closer to being pulled into a Sprint, they need to become more refined. In other words, they need to become more clearly defined so that the team understands what value to produce. The Product Owner publishes a list of stories that he would like to talk about before the meeting so that the team has a chance to look them over. I like for the team to have at least 24 hours, but that isn't a hard–and-fast rule. Technically, the team should have a pretty good idea of what is coming up if they keep an eye on how stories are being prioritized in the Release Backlog.

The Product Owner starts the meeting by introducing the highest-priority story and detailing what the "ask" is—why the customer wants this functionality, and how will it be used. The team figures out what will be developed as a result of this story being worked on.

The team asks the Product Owner to further define the "ask." The team needs to look at the story from the customer's perspective, and this is the Product Owner's opportunity to guide the team's perception of the story. The team is trying to conceptualize the story, so the Product Owner needs to "play customer" here and explain why the customer wants the functionality, how it will be used, and what requirements are important to keep in mind. The key is to not get overly technical. There needs to be enough discussion to get the story properly understood and sized, not to design the functionality up front.

The team should strive to have two or three Sprints' worth of stories refined. Remember, a Scrum Team should expect change. There is no reason to have three hundred stories refined—unless the team can get that done in three Sprints. If the team expects change, it expects priorities to change radically from Sprint to Sprint.

Sprint Planning

Sprint Planning meetings are where the Product Owner explains the Sprint Goal and highest-priority features to the team. The team turns user stories into a detailed list of work tasks. This is where the Product Owner plays the role of customer. The team asks questions about the stories the PO would like to see in the Sprint, and the PO answers them from the customer's perspective.

Try not to just create a task for development, a task for QA, and a task for documentation.

For example, let's say we are discussing a story about creating a powerlifting belt (Figure 5-12). Yes, I understand that a powerlifting belt has nothing to do with software. Work with me here.

Figure 5-12. My weightlifting belt. The weight would fold me in half if I wasn't wearing it

The story's canonical statement would look something like this:

Dave, a super-heavyweight power lifter, would like a heavy belt to protect his back so that he can compete in powerlifting competitions without injuring himself.

Acceptance Criteria: The belt should be made from leather or suede. It needs to be durable enough to withstand the rigors of heavy training, but comfortable enough to be worn for all three powerlifting events (squat, bench press, and deadlift).

During refinement, the team asks the Product Owner to better define the customer need. The PO (speaking from the perspective of the customer) says that powerlifters rely on a thick, sturdy belt to protect their backs when lifting. Before a lift, they cinch their belt as tight as possible and brace their abdominal muscles against the belt. The inter-muscular pressure from this act helps to stabilize their core and keep them upright during the lift.

After discussing the story a bit more with the PO, the team sizes the story at 10 story points.

Now the team is looking to pull the story into a Sprint. As the team begins to think about decomposing the story into tasks, certain pieces of information will come up, and the team may need to think about the tasks that need to be created. For example, the team may ask which powerlifting organization Dave will lift in. There are several powerlifting organizations in the United States. It would be nice if our belt could meet specifications and be legal in all organizations, but that has already been determined not to be possible. The team needs to make sure that the belt is legal in the organization that the customer prefers.

The PO lets the team know which organization the customer prefers, and the team creates a task to make sure that the belt meets specifications. A developer puts their name on the task and estimates it will take seven hours. The developer figures that they will need to get a copy of the powerlifting organization rules, document them in a team wiki page, and let the rest of the team know how the belt should be built. Next, a QA task is created to ensure that once the belt is created, it meets the specifications determined in the previous task. A task is also created to document the specifications so that there is no question that the belt is legal in the preferred organization.

As the team discusses tasks, some design elements will emerge. For example, the PO mentions that while it is not an absolute need, the customer would like the belt to be single-prong. Usually, a lifter will need one or two people to pull on the belt so that it is as tight as possible. It is much easier to get the belt buckled with one prong instead of two. A task is added to create a single-prong buckle, as well as a corresponding QA task to make sure it works correctly.

Each task is assigned to someone on the team and estimated in hours. Once the team is happy that the story is properly divided into tasks, the same routine is done to the next story. To be clear, that's the next-highest story on the Product Owner's prioritized Backlog. There may be reasons why the team can't pull stories in priority order. For example, there may be a story further down the list that needs to be completed to fully complete the story the team just agreed to work on. The team may not have enough capacity remaining in the Sprint to take in the next story; however, they may have room for a smaller story that is further down the priority list. If the PO agrees, they may take that story in next. Whenever possible, the team needs to respect the PO's assigned priority. The team needs to be focused on meeting the Sprint Goal as defined by the PO.

When preparing for the Sprint Planning meeting, the Scrum Master should find out the team's availability for the upcoming Sprint. A normal two-week Sprint has nine working days or 45 available hours per person (ten five-hour ideal days minus one day for end-of-Sprint demo and Sprint

Planning meeting). We are being as transparent as possible, so account for any holidays, planned vacation, or anything that would prohibit work from getting done. For example, a six-person team with no vacation or holidays would have a 270 hour "bucket" for an upcoming two week Sprint.

We then "fill the bucket" as a team. As team members agree to work on tasks, the estimated hours decrease the available hours in both the team and individual buckets. It is important to keep an eye on every team member's individual capacity as well as team capacity. Neither the team nor any individual should be overloaded for the Sprint. For example, if the tester is almost at capacity, but there is still room for other people to commit to more work, someone else on the team should agree to take the remaining testing-related tasks. Of course, a true cross-functional team does not have defined roles so individual buckets don't really have to be maintained. If the team still has defined roles, it's a good idea to keep track of individual capacity.

That brings me to an important point. The team commits to what it feels can be completed in a Sprint. Estimation is not done as an hour-tracking system for management. It is done to ensure that the team has a realistic opportunity to complete all of the stories pulled into a Sprint.

The Sprint Retrospective

Sometimes, when a team is struggling, they will hold a closed-door meeting. It doesn't matter what sport. It could be football, basketball, hockey, or soccer. Any team sport can do it. What happens is that the team holds a meeting just for team members, usually in the locker room. No coaches or management or press are invited. The team uses a closed-door meeting to identify what needs to be addressed and how the team plans to address them.

I look at the Sprint Retrospective meeting as a closed-door, locker room meeting for the Scrum team. It is where the "inspect and adapt" happens. There are many ways to run a Retrospective. I will detail some of my favorites later in this book. For now, I'll detail the basics.

The development team and Scrum Master attend the meeting. Some folks argue that the Product Owner is part of the team and should be present as well. That's a team decision. If the team is comfortable with the PO being there, that's fine. If the PO's presence causes any team member not to fully participate in the retrospective, the PO should not take part.

The meeting must not be a complaint session. The Scrum Master must ensure that there is positivity in the meeting. Allowing the team to go negative doesn't solve anything. Yes, they may need time to vent; however, the purpose of the Sprint Retrospective is to inspect and adapt. Limit the amount of bad stuff discussed, highlight the positive, and decide what to change or experiment with to get better.

My favorite technique to use for a retrospective meeting is the "sailboat" (Figure 5-13). The Scrum Master draws a picture of a sailboat, and gives the team sticky notes. The team is asked to put their thoughts on the sticky notes and to attach them on the picture in one of three places:

- Good things go on the sails (the wind in our sails).
- Bad things go on the anchor (the anchor holds us back).
- Things the team wants to try to do to improve goes out in front of the boat. (Where are we going?)

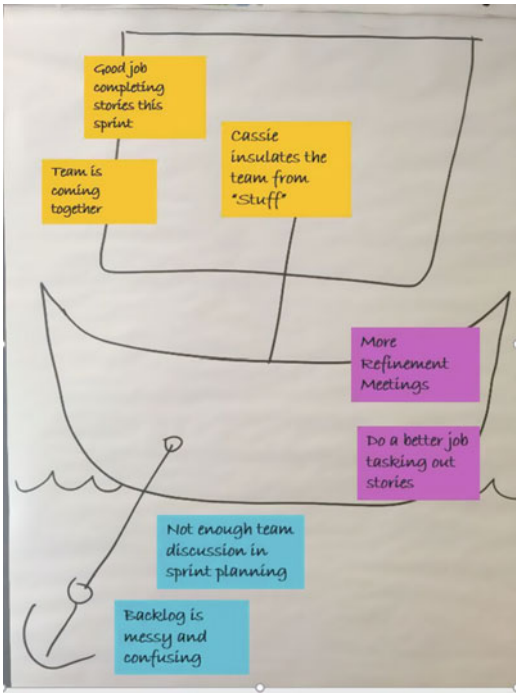

Figure 5-13. The sailboat

The Scrum Master encourages every team member to put something on the boat, while ensuring that nobody dominates the exercise. Usually the suggestion is that everybody must put at least one note on the boat, but no more than three.

Once the notes are on the boat, the team has a discussion about them. Using the fist-of-five technique, the team should decide on action items. The action items mostly focus on the stuff in the front of the boat, but sometimes items will emerge as the team discusses the notes in the sails and on the anchor.

The goal of the Retrospective Meeting is to come out of it with one or two things the team will change or work on in the next iteration. These are the action items. Don't try to "boil the ocean" or fix everything at once. Little changes accumulate and turn into big increases in agility and speed.

At the conclusion of the meeting, the results should be posted in a public place so the team is constantly reminded of the agreed-upon action items. If anybody on the team is uncomfortable with anything posted, do not display that item.

At the beginning of the next retrospective, the team reviews the action items and discusses what worked, what didn't, and how to move forward.

This chapter talked about the Agile artifacts and why they are important indicators to the overall health of an Agile project.

PART

II

Scrum: The Scrum Master's Perspective

CHAPTER

6

The Scrum Master's Responsibilities and Core Functions

At this point, anybody reading this book should have a good idea of what Scrum is. Now I'd like to do something a bit different. I'm going to show you what Scrum looks like by using a Scrum Master by the name of Cassie. No, Cassie is not a real person. I am going to use this mythical Scrum Master and some scenarios that I have either experienced or heard about from some of my peers to show you the Scrum Master's perspective of Scrum.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_6

PICKING THE TEAM

"Hi Cassie—c'mon in."

Duane sat at his desk, surrounded by books about the Agile framework and Japanese armor. Truth be told, he didn't really care that much about Japanese armor. He'd been serving in the role of Agile coach for a medium-sized software development company for a couple of years now. He asked Cassie to serve as Scrum Master for a newly formed team.

"So how was this team put together?" asked Cassie.

Duane smiled and kicked back in his chair. "As I'm sure you know, you never get to hand-pick your team. I am very happy how things turned out with this team. You want the team to be multi-disciplinary with all competencies required to deliver every Sprint properly represented."

Cassie chuckled "Why so many fifty-cent words Duane? Speak English."

"Cassie, have you heard the old adage about building a Scrum team?"

"What's that, Duane?"

Duane smiled and put his hands on his desk. "That a good Scrum team needs to have all of the necessary skills to produce working software every increment, and small enough to be fed with two large pizzas."

"Two pizzas, Duane. Really?"

"Yup… the Scrum team consists of seven people. Kevin is the tech lead. He is the rock star of the group. Jack and Steven are the other experienced developers on the project. Mike and Kelly are new to the company and are relatively inexperienced. From what I understand, they have the chops. They just need time in the trenches. Sue and Ray are the testers. Ganesh handles the documentation."

"Wow, a living, breathing Scrum team. And they're all mine," said Cassie sarcastically.

Duane ignored her and continued. "Annie is the Product Owner. She has been working on a Backlog for a few weeks now. I will warn you, she is a bit rough. Sometimes I think she believes she is the development manager, not the PO."

"So when do I start working with them? Cassie asked.

"Now," replied Duane. "I know you are going to do a great job with these guys. There is no doubt in my mind that you will transform this group of individuals into a high-performing team."

The truth is that a team needs to have the skills to be able to deliver working software each and every Sprint. A Scrum team needs to be cross-functional and self-directed. However, don't get caught up in the myth that everybody on a Scrum team is equal. A Scrum team is a group of individuals working together to deliver what has been agreed upon. The team needs to leverage the skills they have or acquire the necessary skills needed to deliver what is required.

The key is that you want to try to keep the team together as much as possible so that they can learn and grow together. Harmony is created when everybody on the team knows his or her job and how it fits into the team dynamic.

SETTING THE STAGE

"Why are we doing this?"

Cassie had scheduled a team-building exercise. One of the participants blurted that question out and gave a sheepish grin when she looked to see where it had come from.

"Work with me here," she shot back. "This will make sense in a minute or two."

When starting with a new team, Cassie likes to bring them together and ask them to tell her about a team that they admire. It could be a team that won multiple championships or had to overcome great adversity to win a big game. It need not even be a sports team. She's looking for a high-performing team.

"OK, does everybody have a team in mind?" Cassie asked.

"Sure do!" said Mike. "My high school football team won states."

Cassie smiled "What made them so great? What were the characteristics of that team?" She walked over to a flip chart and grabbed a marker.

"Matt, our quarterback, was a great leader." Mike explained. "He never wanted to lose at anything. That guy was the most tenacious human being I've ever come across in my life. His never–give-up attitude was infectious. I would have run through a wall for that guy."

"So, having a great leader is important?" Cassie knew that having a leader was important for team dynamics, but she wanted the team to state this fact.

Steve chimed in "Not necessarily a quarterback that controls who gets the ball, but a leader that inspires the team. No team is going to accomplish anything great without somebody leading the way."

"Great!" said Cassie. She turned and wrote "Good leader" on the flip chart. She turned back to the team and said "What else?"

Kelly jumped in and said "What about accountability?"

Cassie smirked and shot back "What about accountability? What do you mean?"

"I mean, I guess everybody on the team needs to be accountable to each other. We need to hold each other accountable."

At the end of the exercise, Cassie stood in front of a list that looked something like this:

- Good leader
- Hold each other accountable
- Pride
- Great execution
- Good communication
- Trust each other
- Totally transparent

"Congratulations!" Cassie said, "You have just created a set of characteristics that we, as a team, need to strive to emulate."

TEAM RULES

"Let's talk about how this team will work together." Cassie was pleased that the team seemed to be engaged in the meeting and wanted to ensure that she took advantage of it.

"Isn't this kindergarten stuff?" Kevin looked to be a bit perturbed by Cassie bringing the subject up. "Aren't we all adults here?"

"I hope so," said Cassie, "but even adults need guidelines."

Ganesh spoke up. "I'd agree that we should know better, but I guarantee that this team needs one."

"What is that supposed to mean?" Kevin shot back.

"Let's say that you are always late for meetings," Ganesh playfully said. "I mean you are never on time."

"OK" said Kevin, rolling his eyes.

"If there is no team agreement in place, it's difficult to get you to show up to the meetings on time." said Ganesh.

"I get it," said Kevin. You can't enforce rules that aren't in place. "

"Right" said Ganesh. He smiled and said, "The team should agree on team rules and punishments."

"Punishment?" asked a confused Kevin.

Cassie thought it best to interject herself at this point "Yes, punishment. The team should come up with some sort of punishment for breaking team rules. Nothing nasty… buying bagels for the team or something like that." Cassie walked to the flip chart at the front of the room and wrote the word "responsibility" in big letters. "Having a team agreement in place creates shared responsibility," said Cassie. "They are the rules that the team will follow to make yourselves more efficient, and successful."

The team discussed what they thought an acceptable list of team rules should look like. After some back and forth, and a few "fist of five" votes, the flip chart had a list that looked like this:

Team Rules

- Communicate/have discussions/try to get everyone's point of view.
- Get stories done on time.
- Listening—everybody deserves to be listened to. Don't talk over anybody else.
- Team voting.
- Show appreciation.
- Collaborate.
- Value each other's talent equally.
- Constructive debates.
- Like one another.
- Trust and dependability.

Like Cassie, I like to run exercises like this as a sort of team-building workshop. By making the team members focus on a team they each admire, I am actually letting them tell me what a high-performing team looks like. Now the goal is one that they collectively set, not something a manager or the Scrum Master has forced on them. As the team works its way through Sprints, I'll remind them what their collective vision of a high-performing team looks like, and ask what needs to be done to make the current team look like the high-performing team.

Team rules, also known as working agreements, are exactly what they sound like. You set rules, agree on them, and expect everybody to respect them. This is where fun punishments are powerful. Trust me, there is nothing like reminding a tech lead that he needs to buy the donuts because he thought he was above the rules. Team rules create the sense of a level field. We are all in this together.

SPRINT PLANNING

The team was working through a Sprint Planning meeting and Cassie was pleased with the way things were shaping up. Annie had clearly articulated the Sprint goals. Now it was time for Cassie to speak.

"OK, Team, let's figure out our capacity for the Sprint."

"What do you mean by capacity?" asked Kelly. "Didn't we size these stories in our refinement meeting?"

"Yes," said Cassie, "but now we are talking about estimation, not sizing. Let me take a minute and talk about what we are doing in this meeting." Cassie walked to the front of the room. She felt that in order for this meeting to be a success, she should ensure that everybody has a clear understanding of what Sprint Planning encompassed." The goal of Sprint Planning is to agree what stories will be completed in this Sprint. In order to do this, we need to figure out our team capacity and lock it in."

"OK, but what exactly is capacity?" asked Kelly.

"As a team we look at how many working days we have available this Sprint. For a two-week Sprint, that is usually ten days."

"So, eighty hours per team member per Sprint" said Kevin.

"No." Cassie crossed her arms, and then quickly uncrossed them to not look defensive. "I'm talking about actual working time, as in time typing at your keyboard. Taking time out for email, coffee breaks, meetings, and distractions, you realistically should plan for four to five hours a day."

"And take out any days we plan to not be here," added Ganesh. "I'm planning on taking a couple of days off next week to go fishing."

"That's right," said Cassie. "We need a realistic idea of how many hours of team availability we have for the next two weeks."

"So we figure out how many hours we have available. Then what?" asked Kelly.

Cassie smiled and said, "Then we decompose the stories into tasks, estimate how long those tasks will take, and bind team members to those tasks."

"So we had better have a crystal-clear understanding of what we need to do" said Kevin.

"That's right! The team needs to have a good understanding of the story's acceptance criteria and scope," said Cassie. The team will grab a story; remember that we are trying to satisfy Annie's Sprint goals here. The team then will discuss the story and decompose it into tasks. You guys choose the detail you need here. Do what makes sense here. Don't get so detailed that you spend more time updating tasks than writing code. You do need a level of detail that reflects how the work will flow."

"Can you give us an example?" asked Ganesh.

"Instead of one task that says Development, try to create a task for each logical chunk of the development process. The testers can then mirror the development tasks with QA tasks so they can start writing and completing tests for each chunk of development being done."

"Not exactly test-driven development, but it does allow for the testers to get involved early in the Sprint," said Ganesh.

Cassie interrupted: "And allows for greater transparency into what is being worked on in the Sprint." Cassie sat down and said, "I'd also like to guard against optimism here. Be realistic in your estimates. Whatever you do, don't overcommit."

"Cassie, I'm unsure about what to do about tasking out a story. Can you give us an overview?" asked Ray.

"Absolutely!" said Cassie. She was happy that somebody asked for an overview of how to create tasks. This would flow better than if she had brought it up. "Everybody on the team participates. Sure, Kevin, Jack, and Steven have the most expertise, but the whole team is responsible for identifying tasks. I suggest that you account for testing—especially automated test plans. Automation and continuous integration are critical to Scrum. Working software is worthless if it can't be delivered quickly. In order to cut the time it takes to deliver working software, we need to be able to test it thoroughly but quickly. The hidden beauty of test automation is that automated test plans can be written as the code is being written."

"Wow, I never thought of that," said Sue.

"That's right," Cassie replied. "I hate it when the testers get slammed with work at the end of a Sprint. This is a way to help remedy that. Something else to think about is the end-of-Sprint demo. Always be thinking about that and allow some capacity."

"So we are trying to be as transparent as possible about the work to be done," said Steve.

"Yes, and just for you, Steve—don't get too techie. You should be talking about the code at a high level. Don't get into the weeds until the actual work starts." Cassie looked over the room and said, "Make sure your tasks will satisfy the Definition of Done. Remember, if the story doesn't satisfy the DoD, it's not done.

Sprint Planning is where real people commit to real work. After all of the talk about generality when story sizing, Sprint Planning is all about being very specific about what you plan to do and how long it will take. It can take a while to complete a Sprint Planning meeting—anywhere from four to six hours, depending on the maturity of the team. I won't lie; at first, these meetings will resemble torture. Team members will not come prepared, and it will take a lot longer to task stories out than you think. Just as I do with

my cross-country teams, I don't get preachy and tell team members that they need to show up ready to decompose these stories and estimate tasks. After a few Sprints, they will get it.

CREATING THE DEFINITION OF DONE

Cassie walked into the room with her pile of sticky notes and pens, all the while displaying a confident look on her face that masked the fact that she was terrified of what was about to happen. The team was assembled around a large table. Their facial expressions varied from excitement to boredom.

"So, what does it mean to be done?" Cassie asked.

"It means done." replied a team member.

"Ok, what does it mean to be… *done* done?"

Mike, one of the junior developers, spoke up: "Strange game, but I'll play…How about code complete?"

Ganesh smirked and said, "Does code complete mean that the code is commented and cleaned up, or that it just compiles?" He had been around the software business for twenty years and understood the development mindset even though he was primarily writing documentation for this team. "If you guys get your way, you'll pump out so much code that we'll never get it tested and documented."

Mike jabbed back, "Ganesh, we all know that customers don't read the doc anyway. They buy the code that we write…"

"Don't make me put you two in time-out." Cassie joked. "I'd like to remind everybody in the room here is on the same team. I need you all to put whatever preconceived notions you have about roles and responsibilities aside for now and think about what done means for this team." She narrowed her gaze and leaned across the table to make her point. "For the record, done means that you as a *team* have delivered a working piece of functionality. Not code that compiles or looks pretty, but working, tested, documented, functional software. Customers don't buy code—they buy software that helps them solve the complex problems that arise from their day-to-day work." She looked down and noticed that her arms were shaking. She quickly crossed her arms in front of her. She wanted to project authority—especially since she was scared out of her wits. This was the first time she had ever challenged the team, and she wasn't sure what the outcome would be.

After what seemed like an eternity of uncomfortable silence, Kevin (the most experienced developer on the team) spoke up and said, "Alright, Miss Scrum Master, what would you like us to do?"

"You all have sticky notes and pens," said a relieved Cassie. "I want everybody to write something that they would like to see become a checklist item on our Definition of Done and stick it to the flip chart located at the front of the room."

The team started writing their ideas down and something magical happened.

They started talking to each other….

After every team member attached a sticky note to the flip chart, Cassie led a discussion and fist-of-five vote about each one. The team then went through another round of writing ideas on sticky notes and adding them to the flip chart. At the end of the meeting, they had something they could all work with: a simple Definition of Done.

My nephew Max was never what I would consider "food-driven." He simply isn't that interested in food, which is something I can't wrap my mind around. I wake up in the morning thinking about what I'm going to eat that day. Once I eat, I'm already thinking of my next meal. Max is wired differently. When he was very young, all I can remember was him standing up in his high chair and screaming, "Done!"

Of course, he wasn't done. His mother would say something sweet like "Honey, eat just a little more."

"Done!"

"But you haven't finished your…"

"Done!"

Max screaming that he's done when *he* thinks he's done has a lot more to do with how development teams act than I'd really like to admit. If allowed to operate unchecked, all kinds of stuff gets "back-burnered." You know, pushed out of the way to be finished later so the team can get the big, important stuff done.

It's done with the best of intentions. The team really does plan to get to all of that back-burnered work, but the reality is that there is always more work than the team will have time to complete. That back-burnered work really isn't insignificant. It may not be creating features, but it includes stuff like thorough testing and possibly infrastructure. This usually becomes technical debt. Debt that will have to be repaid in the future.

The Definition of Done is critical to the success of a Scrum team because it forces the team to ensure that nothing has been forgotten or pushed to a later time. When the team works to satisfy the DoD, everybody from the Product Owner to stakeholders can have confidence that the team is producing potentially shippable functionality. Some teams use the DoD to create tasks for stories.

Nothing can be done until the thing is built, tested, documented, and demonstrable. If the team works to satisfy the Definition of Done, everybody understands that this is the case. There is no need to question that the story is completed. If it's not complete, it's not done. Follow through… Finish!

DEFINITION OF READY

"OK, I think I understand the concept of the Definition of Done, but what is the Definition of Ready?" Steve asked.

Cassie chuckled under her breath. The Definition of Done and Definition of Ready are similar conceptually, but very different in practice. "Well, Steve" she said, "If you were going to go skydiving, would you want to make sure that you double-checked that you had a primary and backup parachute?"

Steve scratched his head and said, "Well, yeah… I would make sure that I had a parachute, for sure."

Jack jumped into the conversation: "Not only do you want to make sure you have both parachutes, you want to make sure that they were packed correctly." He rolled his eyes and said, "The last thing you need is two parachutes that don't work."

Kelly added, "You also need one of those things they wear on their wrist. What are they called, an altimeter or something like that?"

"So you know when to pull the ripcord," said Cassie. "What you guys are describing here is a Definition of Ready. Just like you wouldn't want to jump out of an airplane without making sure everything is ready to get you back to terra firma safely, you want to make sure that a story is ready to be pulled into a Sprint before doing so."

My wife and I run a youth program at our church. We primarily work with the younger children, mostly because they think my jokes are funny. Normally, at some point in the evening, we have some sort of relay race game. It gives the children an outlet for their boundless energy and is a lot of fun for both the kids and adults. I usually start the race like this:

"Ready?"

"Set?"

"Don't go yet!"

Everybody sprints for three or four steps. Once they realize what I've done, there is usually a bunch of giggling and mock protests about my false start. We get everybody back in place and I try to restart the race…

"Ready?"

"Set?"

"Don't go yet!"

Once again, the children are so excited about starting the race that they don't listen, and start running. I usually go through this process three or four times before the kids catch on to my foolishness.

I see the same things with Scrum teams. Not the whole dealing with my foolishness thing, but the fact that they are so concerned with getting started that they forget to make sure that a story is ready to start. It is not a good feeling when a story is pulled into a Sprint and a week later, you realize that a prerequisite task was not completed. Having a Definition of Ready in place guards against stories being worked on before they are ready. It makes sure you can ready, set, GO with a user story.

STORIES

"Hi Annie, how are you today? Are you ready to look at some stories?" Cassie was upbeat and looking forward to working with the Product Owner.

"Hi Cassie," said Annie. "I wish I could tell you that I was ready to work on stories, but I'm frustrated at the moment."

"Uh oh… nobody likes a frustrated PO." Cassie slid her glasses back up to where they belonged on her nose. "What are your pain points?"

"I don't think the team understands what I'm asking them to give me. I mean, they are smart people. I like to think that I'm a smart person, and the customers have clearly explained what they want to me. When we all get in a room and I try to explain it to the team… it's like we are all missing something." Annie threw her head into her hands. "We start talking about a story and it takes hours to get them to understand what I need them to build. If I can get them to understand at all."

Cassie tilted her head and smiled. "Is that all that's frustrating you?"

"Well, yeah," said Annie in a bit of a startled tone.

"It just so happens that story writing is something I'm really good at, so let's get started."

"I don't know what kind of voodoo you have working over there, but you have my attention."

Cassie giggled, "It's not voodoo, just good story writing practices." Cassie sat down and folded her hands in front of her. "Who writes the stories?"

"Well, I do," said Annie.

"Annie, you mean to tell me that you write all of the stories?"

"Yes, I do.

"Why?"

"Because I don't want a bunch of junk cluttering up the Backlog."

"Hmmm." Cassie tried very hard to put a serious look on her face. "You are correct… kind of. The Product Owner is ultimately responsible for Backlog content, but anybody

should be able to write stories. You shouldn't worry so much about who does the writing. We need to work as a team. To help each other out. You absolutely do not want junk in your Backlog. That doesn't mean that you have to write anything. It means you have control of what is accepted into the Backlog."

"I really don't see what the big deal is here." said Annie. "Who cares if I write all of the stories?"

"OK, let me put it this way," said Cassie. "What if one of the team members told you it wasn't their job to write stories."

"You mean that they would literally give me the 'it's not my job' line?" Annie started making the quotation sign with her fingers.

"Yup."

"My goodness, that's sounds horrible." Annie puts her hand to her mouth and had a shocked expression on her face.

Cassie leaned forward and said, "Truth be told, it's disrespectful. Disrespectful toward YOUR team. In my opinion, stories turn out best when they are written collaboratively."

"I never thought of it that way. I guess I need to tell the team that I need help writing stories."

"You go, girl!" Cassie put her hand up for a high five, but pulled it back when Annie gave her an uncomfortable look. "OK, let's talk about how some of your stories are written."

"I can hardly wait," replied Annie.

"Let's take a look at the highest–priority story in your Backlog. The title is 'Create an SSL Connection to a Select Server.'" Cassie slid her glasses back down her nose and looked at Annie over them. "Do you know what that tells me?"

"What?"

"Nothing…"

"Oh, come on!" Annie exclaimed. "I'm telling you exactly what you are getting."

"Not really," said Annie. "The title of a story needs to be concise but at the same time communicate customer value. As a customer, I really don't care if you are using SSL, or any technology for that matter."

"All you care about is that we provide a secure connection to the server?" asked Annie.

"Bingo! There is your story title. This way, anybody—a customer, internal stakeholder, team member, or even the CEO of the company can understand what value this story will produce. You also want the title to be easily searchable."

"So that I can find it easily?" asked Annie.

"Right. It doesn't hurt to think about that kind of stuff up front." Cassie went a bit deeper into the story by saying, "Next we have the canonical statement."

"I need a bit of help with that one," said Annie. I don't understand why it's necessary to write the description in this format."

Cassie sat back in her chair. "You are describing who the story is for, what the story describes, and most importantly what benefit is attained for the user by adding the value you just described. Remember, you are trying to get the team to look at this through the eyes of the customer. Using the canonical statement makes it easy to do this. It communicates who the change is for, what changes, and why the change is a good thing. Does that make sense?"

"I guess," said Annie.

Cassie folded her hands in front of her and said, "My challenge to you is that you don't say how you want it. You say what you want."

Annie's eyes lit up. "Because the team decides how they deliver the value. I just have to be clear about what I'm asking for."

"That's correct! You are starting to get it here. Now let's talk about personas."

"Personas? What are they?" asked Annie.

"Everything we do has someone who consumes the result. Think of a character who embodies your 'user' and give them a name. That's a persona."

Annie scratched her head and said, "Make up a fake person?"

"That's right!" Cassie answered. "We need the team to look at stories not only from the customer's perspective, but from the *correct* customer's perspective. You know—the customer who will actually use our product."

Annie thought about what Cassie had just said for a minute and said, "So, by playing make–believe, we actually get the team to think like the customer and build something they want to use."

"I like to say build something the customer can't wait to get their hands on," Cassie giggled. She was happy that Annie was starting to see the value of personas. "Can you see how personas can be a powerful tool?"

"Oh yes," exclaimed Annie. "I'd like to sit down with the team and develop some personas right away."

Writing user stories can be a difficult skill to master. Stories need to be small enough to fit in a Sprint but contain enough value to stand on their own. The idea is to get the Scrum team to talk about the requirements (both functional and nonfunctional) necessary to create a slice of value that can be installed. Most people I talk to describe user stories as small chunks of functionality. I prefer to call then units of deliverable functionality. Most teams that I have worked with can get to the point where they can produce something they can demo every two weeks. The real goal is to have functionality that is ready to

demo and install at the end of every Sprint. This can be a game-changer for teams, as they need to rethink how they develop software. Teams often need to revisit how they write, source, control and build code in order to be able to deliver on this cadence. It also becomes evident to the testers that test automation is critical. Testing needs to happen as early in the Sprint as possible, and test cases have to be automated as much as possible.

All of that needs to be reflected in the user stories.

Stories always need to be written from the customer's point of view. I like developers. I don't want anybody thinking that I have a chip on my shoulder about Scrum team behavior, but I know this to be a fact. The second I let a Scrum team lose customer focus, they start talking about coding details.

Personas are a powerful tool that the Scrum Master and Product Owner can use to keep the team focused on the customer's perspective. I will admit that the whole concept looks silly at first. Personas are made-up users of specific types that will help the team develop software that isn't… generic. At the risk of being simplistic, you create a bunch of imaginary users and write software for them. These personas represent the characteristics of actual users, so they aren't totally made up.

Yes, I'm serious.

The software that Scrum teams produce is normally used by more than one category of person. Using personas allows us to look at what we are building from the user's perspective. They force us to embrace a more user-centered approach to how we do things—to put the user first. By understanding how a given feature affects Bob the operator, Jerry the automation engineer, and Fred the MVS sysprog, we can actually do a better job of understanding the consequences of design choices. Personas will help the Scrum team to better understand what value the stories they are working on produce.

BACKLOG REFINEMENT

Annie and the team gathered in a conference room for a Backlog Refinement meeting. Cassie, upbeat as usual, asked the room "Is everybody familiar with what a Refinement meeting is? In other words, do you know why you all are here?"

Ganesh spoke up "we are getting stories ready… right?"

"That's right, Ganesh" said Cassie. One of the things we are looking to do here is to satisfy the Definition of Ready so that a story can be pulled into a Sprint. What else?"

"I don't know," said Ganesh.

"OK, does anybody else have any ideas?" asked Cassie.

"You want to torture us with more meetings?" asked Ray.

Cassie made a mental note about the fact that the usually quiet Ray spoke up. She was slightly alarmed by the fact that he wasn't seeing the value of the Refinement meeting. "Ray, I can understand why you may think this is just another meeting, but you and the team need to understand that the Refinement meeting is important and even valuable. This is where the Product Owner introduces stories that could be worked on in the next two or three Sprints. As Ganesh said, the Definition of Ready must be satisfied; however, other things happen in the Refinement meeting. You guys need to refine the story's acceptance criteria so that everybody is clear about what is to be delivered. The story must be estimated."

"That means we assign story points," said Kelly.

"That's right. The team needs to size the story with an initial estimate. Before that happens, you guys need to figure out the scope of the story."

"Who exactly is 'you guys'?" asked Jack, making the air quotes gesture with the index and middle fingers of both of his hands.

Cassie shot back, "The team and the Product Owner. It makes sense to understand the scope of the story before you estimate it. The story may need to be split. The story may need to be refactored. Annie may decide that this particular story is too expensive and move it down in priority."

"Whoa, what does that mean?" asked Sue.

"It means that I may prioritize stories with smaller scope over this one." Said Annie. "We want to deliver the most value possible. Sometimes it makes sense for the business to get a bunch of smaller stories done."

"Instead of something that is big, bloated, and costly," Said Steve.

Refinement meetings are valuable. They are where the rubber meets the road. Where the team and Product Owner get together and work stories out. Typically, Backlog Refinement should happen every week and take one to two hours. The purpose of the meeting is to introduce stories that are candidates for the next few Sprints. When everybody gets together and starts talking about a particular story, a couple of things will become clear. First, is the story too big? No story should take more than a quarter of a team's Sprint velocity. That is a rule I coach teams to live by. A Scrum team should be striving to make stories as small as possible. Small stories have big benefits:

- They remove variability. OK, there will always be variables that arise during a Sprint. Keeping stories small helps to control and deal with variability.
- They are easier to estimate.
- Small stories are easier to understand.
- It is easier to write acceptance criteria for smaller stories.

Try to keep user stories as small as possible, but not so small that they are actually tasks. Finding a balance will be uncomfortable at first. It probably requires the whole team to collaborate to achieve the goal. The team may need to get creative, but when they are motivated it can be done. Any task can be broken down into smaller steps that are more manageable. I don't care if you are talking about business or weight loss or learning how to play the guitar. If you want to lose 100 pounds, take it meal-by-meal, workout-by-workout. Focus on the task in front of you and do your very best at it.

In the end, it really isn't the team's fault. Waterfall encourages a team to hide the bad stuff. Scrum will expose the bad stuff… quickly. The beauty of this is that once the bad stuff is exposed, the team can easily remove it because following Agile methodology requires a constant striving to get better.

Learn the discipline to break stories down and finish in Sprints.

ACCEPTANCE CRITERIA

"You need to think of acceptance criteria as a contract between you, the PO, and the team about the work to be done."

"A contract? Isn't that a bit rigid?" asked Annie. "Isn't that a bit much?"

Cassie replied "No. Acceptance criteria is the best way for you to ensure that the team understands what you expect them to deliver."

Annie's face lit up and she said, "So we can eliminate the 'Oh—you wanted that, too' response from the team."

"That's right!" said Cassie. "Good acceptance criteria gives both you and the team a complete understanding of what success looks like. After reading the acceptance criteria, the team should have a firm grip on the story objectives. It should detail some use cases that are within the scope of the story, as well as those that are not."

"How should I do that?"

Cassie could sense the proverbial light bulb go on at this point. "You need to focus on the correct or expected outcome. It could be as simple as commands and their desired output or complex use cases that show the desired functionality. The bottom line is that you need to assume that if it's not there, you are not going to get it."

Annie chimed in. "So this is my opportunity to detail exactly what I want."

"And ensure that everybody truly understands what we are trying to do," said Cassie. She continued, "If a story has good acceptance criteria, anybody on the team should be able to look at the story six months later and understand what functionality and value will be derived when the story is implemented."

Good acceptance criteria remove any vagueness or uncertainty from the story. I coach Product Owners and teams to build acceptance criteria using use cases and outcomes. The result should not be a slapped-together list of conditions to be met without specific goals. It should focus on desired outcomes that the user story will produce. If it's not included in the acceptance criteria, then it's not required for story completion.

It's as simple as that.

REQUIREMENTS

> "Requirements go into user stories. That's the way we do it in Scrum." Cassie was addressing some stakeholders and executives who had some questions about how Scrum worked. "Annie, our Product Owner, is responsible for gathering requirements. That's why she is talking to you all and customers to find out what we should be building. What she gathers is turned into user stories—small chunks of value that the team works on during iterations. A user story describes the requirement at a very high level, saying, 'As an X (user), I need Y (functionality) so that I can Z (gain some type of value).' The team and Product Owner refine the story and, eventually, more detail and the acceptance criteria will emerge. The acceptance criteria details the actual requirements—what the team needs to deliver so that Annie can accept the story and demonstrate it at the Sprint Review meeting."

How many times have you said to yourself "this thing would be so much better if it did *this*?"

Requirements are what customers and stakeholders want to see become reality. It's up to the Product Owner and product management to gather these requirements and prioritize them. Admittedly, this can be tricky for a couple of reasons.

Most of the time, customers and stakeholders don't understand what they are asking for.

Requirements change very rapidly.

No, I'm not saying that customers and stakeholders are stupid. Nothing could be further from the truth. What I am saying is that requirements can be very complex and, just like architecture, emerge over time. Adding to this complexity, requirements can change rapidly. That means that what a customer needs today can change radically by next week, or the end of the team's current iteration. Scrum does not require that all requirements be defined up front. Requirements are expected to change as teams iterate. That is one reason why any Agile guru worth his or her salt will tell a Scrum team to keep iterations as short as possible.

ESTIMATION

"OK, team," Cassie said, "We are in a good rhythm here." She was holding a Backlog Refinement session and was pleased with how the team was working. They had just finished discussing the ins and outs of a particular story. Now it was time to size it. She passed out a set of cards to every team member; each card had a number written on it. "We've had some issues with sizing to this point, so I'd like to try something new. I've given you all seven cards. Each card has a number on it. These numbers are called the Fibonacci sequence—one, two, three, five, eight, thirteen, and twenty. I like to use the Fibonacci sequence because the numbers really don't relate to each other, but they do correlate well to t-shirt sizes. Extra small, small, medium, large, extra-large, extra-extra-large, and too big for a Sprint."

"Do we really need all of this, Cassie?" said Kevin. "Isn't a story point just a day of work anyway?"

"Kevin, that will get you into trouble," said Cassie. "Human beings, the species we all belong to, are terrible at estimating how long it will take to do something."

"Ain't that the truth," said Mike. "Every time I do something around the house, I tell my wife it will take fifteen minutes and it ends up taking two hours."

Cassie raised an eyebrow and asked, "Why is that, Mike?"

Mike answered, "Well, I get into something and find out there is more stuff to do what I thought."

"Exactly," exclaimed Cassie. "Humans are terrible at estimating how long it will take to do something, but very good at comparing things."

"How?" Mike was clearly perplexed.

"OK, Mike, using the cards in front of you, what number is an aircraft carrier?"

"Um….Thirteen?" said Mike.

"Wouldn't an aircraft carrier be more like a twenty?" said Kelly. "That ship is huge. Bigger than anything else I can think of."

Mike replied, "OK, twenty it is."

Cassie smirked and said, "If an aircraft carrier is a twenty, what is a houseboat?"

Mike laughed and said "Not a twenty." He rubbed his chin, thought for a second and replied, "I guess a houseboat would be a three."

"There you go," said Cassie. A story point is what you expect your effort to be to complete a story in relation to other stories. Well, effort isn't the best word to use. You should be gauging how much effort is involved along with the complexity of the story and any doubts you may have."

Kevin jumped in: "I see now why points don't necessarily equate to time."

"How so?" asked Cassie.

"Because different people on the team have different skills and experience. What might take me a day might take someone else a lot longer and vice versa."

"Exactly" said Cassie. "Notice that I didn't get specific about what type of aircraft carrier or houseboat. There are varying sizes of each. Estimation is a relative comparison of how much work is needed to complete a story. It has nothing to do with how long it will take an individual team member to complete the story. No matter who works on the story, it has the same story points."

"Cassie, I don't think this team can build an aircraft carrier in one Sprint," said Kevin.

"That's why I picked an aircraft carrier." Cassie was so excited. She had to actually restrain herself from jumping up and down. "A twenty is obviously too big to fit in a Sprint. That story needs to be split up or refined further."

Estimation is not an exact science. I have found it difficult to convince a bunch of people with engineering degrees that something as imprecise as story points could be valuable.

It is.

When done correctly, story points provide a remarkably accurate view of the work remaining to be done and how long it will take the team to complete it. The team just needs to be honest with themselves about the volume of work to be done. It's not just about writing code. It's about everything needed to deliver what the story requires (development, documentation, QA, and so on). Don't just follow the tech lead or architect. Remember the Agile Manifesto here. The more the team discusses the story, the more the team will understand the story.

Agile gurus always suggest the Fibonacci sequence be used when sizing stories. Since four out of the first six numbers are prime numbers, it makes it impossible to break things down into equal chunks and work on them in parallel.

A word about splitting stories. When a story is too big, it is tempting to split the story by tasks. For example, split the story so that the development work goes into the first Sprint and the testing and documentation happen in the second Sprint. This isn't feasible in Scrum, because you have nothing to demonstrate at the end of Sprint one. Remember, we want to build value in demonstrable increments. When a story is split, half of the development, testing, documentation, and anything else required needs to happen in Sprint one. The team now has a tested, demonstrable piece of functionality to show to customers and stakeholders.

And never forget to look at stories from the customer's viewpoint.

VELOCITY

"What are you doing, Cassie?" Sue was a few minutes early for the daily stand-up. When she walked into the room, she found Cassie with a pile of papers in front of her.

"Oh, hi Sue." Cassie sang, "I'm working out the team's average velocity."

Sue made a scrunched up face like she was thinking hard. "Velocity… I've heard that word used in meetings before. What does it mean?"

"You mean to tell me I never explained velocity to the team?" Cassie was slightly perturbed with herself for forgetting to clearly explain something so important. "OK, are you ready to learn the great secret that is Agile velocity?"

"Sure," giggled Sue.

"Velocity is how fast the team is delivering value to its stakeholders."

"So it's how fast we deliver tested code?"

Cassie pushed her glasses down her nose. "No, it's how fast you deliver value. Did you ever wonder why we don't estimate defects?"

"Do you mean the stuff we find wrong when we test?" asked Sue.

"Yes, those are defects. We give them an ultra-high priority because we want to give our customers stuff that is defect-free, but we don't estimate how big a job it is to fix them. Did you ever wonder why?"

"Now that you mention it, we don't estimate defects. Why is that?" asked Sue.

"Because customers expect our products to be defect-free. We aren't really delivering new value when we are fixing defects. We are doing something that the customer expects to be there anyway. Just like when you buy a car you expect it to be in perfect working order… right?" Cassie said.

"Absolutely!" answered Sue.

Cassie smiled and replied, "That's why we don't story-point stuff like defects and technical debt. Those things aren't really delivering value to our customers and stakeholders, because they expect that kind of thing to be done anyway. In other words, they expect our product to be in good working order and defect-free. If we produce something that is full of bugs, it slows our velocity. Same with technical debt. That is the stuff that we allowed to go out into the field because we were trying to make a date. I classify it as the sins of the past. We know it is in the field, just waiting for a customer to find it."

"And we should clean that stuff up. Being a tester, I really don't like releasing anything that doesn't seem right to me. You are preaching to the choir, sister!" exclaimed Sue. "So what are you doing with all of these papers?"

> "The team has gone through three Sprints, so I now have enough information to calculate our average velocity. It's simple, really. Take how many story points the team has completed each Sprint and divide by the number of Sprints. It will show how we are doing as a team, and predict how long it will take us to complete the story points left in the release Backlog."
>
> Sue smiled and said, "So, no more arguing about dates?"
>
> Cassie replied, "We can still argue, just not about when we can release. Our average team velocity shows how quickly we are delivering value."

Bugs are the inevitable truth of software development. No matter how hard you try, things will go wrong—and you will have bugs to fix. In addition, defects are to be dealt with quickly and harshly in Agile.

Which means zero defects… As in no new bugs—Zero, none, zip, nada, nicht.

The idea is that a defect does not increase in value over time, but it does increase in cost, *exponentially*. The standard that I have seen is that a defect should be closed in the Sprint in which it was found. The idea is that we do not, under any circumstances, want to build a mountain of bugs.

OK, that's fine and all, but what do we do with all of the stuff that has been in our products for years? Like the stuff that was "negotiated down" back in the Waterfall days that we never got around to cleaning up?

That, my friends, is technical debt, as I detailed in Chapter 5.Again, it's much more expensive to correct a bug once a product has been released. You need to rip the modules back open, re-learn how the code behaves, and if a customer finds it, move heaven and Earth to get it fixed quickly and keep them happy.

Every product has some measure of technical debt. From hard–to-understand dependencies in stable, rarely modified modules in the dark recesses of your product to sloppy code that is often ignored in addressing higher-priority problems and never revisited—it's there… lurking. Just like defects, technical debt needs to be identified and attacked. You certainly can't give it as high a priority as a defect, but Product Owners should be encouraging their Scrum teams to be taking on a bit of technical debt every Sprint. The number that I've heard is 30% of team capacity.

I'd be remiss if I didn't also say that stuff like writing automated tests is also technical debt. In my mind, technical debt is really anything that does not provide value to the customer. It could be argued that this is good debt. I agree—writing test automation enables future value by allowing faster regression testing. It still delays customer value right now. Technically (sorry for the pun), defects are debt too—we just try to "pay them off" immediately.

This is why both defects and technical debt should not be assigned story points. They should negatively affect velocity and give an accurate picture of how fast the team is getting features done. I like how Bob Carpenter defines velocity. Velocity is how fast a Scrum team delivers value to customers. A customer already expects the product we sell them to be tested, and bug free. The value is in the features we deliver.

Velocity is the number of story points your Scrum team completes during a Sprint. Sprint after Sprint, you develop an average velocity which allows you to better gauge when you will be done with a project.

How can something that I call a "guesstimate" be used to track a project? In other words, how can something that is sized using an inexact methodology be used to give you an exact release date, or even a release date that is even in the same ballpark?

Simple. As long as you uniformly assign story points, they will give you a more exact idea of how long something will take rather than using hours and/or days. Story points give no sense of precision like hours or man-days do. If your story-point estimates are consistent, velocity quickly shows not only how fast a team is moving, but how stuff that is external to the team affects the work that the team can complete in a Sprint.

It's not how many hours you spend working in a Sprint—it is how many story points you complete per Sprint and how many are left to do. This is why you want to complete your stories in the same Sprint where you started them. There are a bunch of reasons for this. You want to size your stories small enough to complete in a Sprint so that you get feedback and deliver value to customers in increments. However, the biggest reason in my mind is that the story points for a particular story are recognized in the Sprint where the story is closed. If you consistently allow stories to span Sprints, you are negatively influencing your team's velocity.

IMPEDIMENTS

"Hi Richard," Cassie sang as she walked into the functional manager's office.

"To what do I owe this privilege?" asked Richard. "A visit from the most positive person I know."

"Oh, stop it," Cassie feigned protest, but was thrilled that somebody noticed her constant positivity. "The team brought up a couple of things at today's stand-up meeting, and I'd like your help in solving them."

"Ok," said Richard. "How can I help?"

Cassie sat down and folded her hands. "The first issue is that we need source control for our automated test plans. I talked to Sue and Ray about it after the meeting…"

"One of your parking lot meetings?" asked Richard.

"Why yes." Cassie was smiling ear to ear at this point, "They said that they would need a server and version-control software. They did mention one that they like, but I can't remember it off the top of my head, and I left my notes at my desk."

"No worries, Cassie. This is something that is in my wheelhouse. I'll get the testers together and take care of what they need. They will probably need some budget allocated, so I'm the right man for the job!" He sat back in his chair and put his hands behind his head. "What was the other issue?"

"A sales rep has been trying to contact the team directly. I'm pretty sure he's either trying to close a deal or do something with a proof of concept," said Cassie.

Richard's eyes narrowed, "Do you think he should be talking to the Product Owner?"

"Right," said Cassie. "We need to keep distractions to a minimum. The team needs to focus on the Sprint goals. This sales opportunity may be big, but it must be vetted with the Product Owner and prioritized."

Richard rubbed his chin and said, "I want to make sure that I'm crystal clear on this. Why does the Product Owner need to be involved?"

"Because the Product Owner is responsible for the project," Cassie asserted. "The PO is the one wringable neck that is responsible for what happens. If this sales thing is truly important enough to require the team's attention, the PO needs to determine what gets removed from the Sprint and sent back to the Backlog. She needs to be able to clearly articulate the work and acceptance criteria to the team so they can focus on the sales opportunity."

"I see," said Richard. "Nothing is free."

"That's right, Richard. You can't make the team work extra hours and weekends unless they agree to do it."

One of the main duties of the Scrum Master is to remove impediments. *Impediments* are defined as anything that can slow the team down or prevent them from doing their jobs. Here is another reason why the daily stand-up meeting is so important. One of the three questions that requires an answer is, "What is blocking you?"

The Scrum Master needs to remember that while it is their responsibility to remove impediments, they don't have to do it alone. The functional manager should be in lock-step on this topic. Remember, the Scrum Master really doesn't have any authority. You know, the whole servant-leader thing. The functional manager does, and can help to get things done.

An impediment could be something like a team member's laptop needing repair. In this case, the Scrum Master would procure a loaner system and make sure that the team member has everything necessary on the loaner laptop. They would also ensure that the broken hardware is fixed. This way, instead of the team member dealing with the help desk, they are working on Sprint goals.

The team also needs protection from outside interruptions. As Cassie said in our story—everything must go through the Product Owner. Whatever is being asked of the team must be prioritized and adjusted for. If the team is expected to drop everything, everyone needs to understand that there are repercussions to this action. You can't just throw things on the pile and expect the team to deal with it. The Scrum team needs to be able to focus on delivering the Sprint goals every iteration.

SPRINT EXECUTION

Cassie set up for the daily stand-up meeting as usual. The Scrum board was prominent in the center of the room. The board clearly showed the status and progress of each story in the Sprint Backlog. She updated the Sprint burn-down chart, and waited for the Scrum team to arrive. The more she looked at the Scrum board, the more she didn't like what she saw.

The team filed in, displaying a wide range of emotion. From joy to grumpiness, the team gathered around the Scrum board and looked at Cassie.

"Should we start?" asked Sue. One of the things Cassie made clear to the team from the very beginning was that this was their meeting. She was not running it, so she wasn't interested in asking any questions. In fact, after a few stand-up meetings Cassie stood and said nothing at the beginning of the meeting. They all stood around in uncomfortable silence until Steve finally started talking about what he did yesterday, what he planned to do today, and any blocks he was experiencing.

"Before you guys start, I'd like to point something out." Cassie hated to disrupt the meeting like this, but she strongly felt that she needed to address the situation at hand. "Do any of you see any potential problems with our Scrum board this morning?"

The team looked puzzled. Mike piped up, "Cassie, I think it looks great. We are getting all kinds of work done on these stories."

"Maybe," said Cassie. "But here is my concern. Every story in the Sprint Backlog is currently in progress."

"How is that a problem?" asked Jack.

"The goal of a Sprint is to get stories done." Cassie walked over and touched the Scrum board to draw attention to it. "Looking at the Sprint burn-down and the Scrum board, do you think we will complete all of these stories by the end of the Sprint?"

"Absolutely," said Jack with a smirk on his face.

Ganesh was a bit more skeptical: "I don't know about that, Jack. There is a ton of work left up there, and we are already halfway through the Sprint."

"We'll get it done, Ganesh. No matter what. We will deliver what we promised."

Cassie saw her opportunity and seized it. "Jack, that might work in the short term, but it's not sustainable long term."

Jack cut Cassie off and said, "I know, the manifesto says the team should be able to work at a pace they can maintain indefinitely, or something like that."

"That is correct, Jack, and the way a Scrum team can attain that pace is by limiting WIP." Cassie wasn't sure, how open the team would be to this concept, but felt it was time to challenge them with it.

"What is WIP?" asked Kelly.

"What Cassie is going to do to us to make us work faster," joked Steve.

Jack didn't say anything. He just stared at Cassie and waited for an explanation.

"WIP stands for work in progress. Let me ask you a simple question." Cassie turned to Mike, who had been quiet to this point. "What is better at the end of a Sprint, having one hundred percent of the stories fifty percent done or fifty percent of the stories one hundred percent done?"

Mike bit his lip and thought about it. "I guess we would want to have fifty percent of the stories done."

"Woop woop! Mike gets the prize!" exclaimed Cassie. "We want, as a team, to get the highest-priority stories done. For example, instead of one person taking a story each and working on it, you only work on the top two highest-priority stories until they are done."

"I don't see the value in this," said Jack. "Sounds like…"

Cassie jumped in. "My turn to cut you off. Jack. What if you were out in the middle of a lake in a boat, and all of a sudden, there's a hole in the boat and water is rushing in. What would you do?"

"Why, I'd start bailing water as fast as I could…"

Cassie smiled and asked, "Shouldn't you plug the hole in the boat first?"

Jack gave a sheepish grin and said, "I guess so."

"When teams don't limit their WIP, it's very easy to forget priorities. All you see is a mountain of work. However, if you swarm the most important stories first, they get done. The mountain gets smaller, and the team moves to the next highest priority."

"But why does it matter?" asked Mike. "We are probably going to get everything done."

Cassie smiled and said, "Maybe. But if you limit WIP as a team, there will be at least a ten percent rise in velocity. I think it has something to do with the psychological effect of seeing things get done. It creates a kind of feeding frenzy or avalanche effect."

"So, I suppose you want us to aggressively limit the number of stories in progress," said Mike with a hint of sarcasm.

"I tell you what," said Cassie, with just enough sass to counteract Mike's poor attempt at being witty. "Why don't you guys just try it? Swarm a couple of stories and see how it works. If we agree that limiting WIP is working out, we can expand the practice."

"Do you mean you want us to do a kind of experiment?" asked Sue.

"That's exactly what I mean," answered Cassie.

Managing work in progress (WIP) is a sign of maturity. For some reason, most people want to start working on everything at once. I call this the curse of multitasking. Yes, we live in a time that seems to demand that we multitask, but in my mind, it's more valuable to get stuff done. Getting stuff done requires that we focus. There is always more work to do than we can get done. Without focus, we get overwhelmed and important things like quality suffer. One of my favorite sayings is "It's better to have fifty percent of your stories one hundred percent done than one hundred percent of your stories fifty percent done." Limiting WIP forces teams out of Waterfall thinking. We don't value being busy in Scrum; we value completing stories and delivering value. Limiting WIP allows a Scrum team to focus on delivering complete, tested stories. It may seem counterintuitive to allow some stories to sit while the team focuses on getting the higher priority done. However, when you think about it, the team is getting the high-priority stuff done. The big secret is that those lower-priority stories will still get done, because focusing on smaller batch sizes allows a team to get stuff done faster. There is power in doing what you say you are going to do. WIP limits guarantee that the team will deliver the high-priority stories every Sprint.

This is why I'm a big advocate of teams doing experiments. In the case of WIP, I'd coach teams to try swarming a couple of stories to see if they can finish them faster. If the team realizes the benefit of the experiment, they will be open to adopt the practice. In this case, set up WIP limits in a broader sense.

In my experience, most teams push back when you try to change something. Change is scary and unknown. That being said, teams are usually open to doing an experiment for a couple of weeks. If things do work out, the team throws the idea away. If value is realized, the team can adopt the practice.

PULLING ADDITIONAL WORK INTO AN ITERATION

"Cassie, can we ask you a question?" said Mike. He was standing by Cassie's cubicle with Kelly in tow.

"Sure, what's on your mind?" said Cassie.

"Kelly and I are having a bit of a disagreement, and we aren't sure who is right," said Mike.

"And we want you to let us know." Injected Kelly.

"Okay, shoot," said Cassie with a smile on her face.

"Basically, I'm done with all of my Sprint work" said Mike. "Can I pull a new story in from the Backlog? I want to get a head start on work for the next Sprint," said Mike.

"I don't think that's what we are supposed to do." said Kelley. "Isn't that against the rules?"

"There aren't any rules," giggled Cassie. Scrum is a framework. The idea is that you guys, the team figure out how to work within the framework's guidelines."

"If there are no rules, why do we have to go to all of these meetings?" asked Kelly.

"You need to operate within the framework. That means that you need to iterate every two to four weeks and include all of the Scrum ceremonies." Cassie paused for a second and bit her lip. "Okay, maybe that stuff could be considered as rules, but we try to keep them as minimal as possible. I don't want to see anybody dictate to the team how to perform the work. Just have what is needed in place to keep everybody aligned to the Sprint goals and project vision." Cassie sensed that the conversation was straying from the original question, so she looked at Mike and asked "So, you are done with all of the work you committed to for this Sprint?"

"Yes" said Mike.

"Is the rest of the team done as well?" asked Cassie.

"Well… no," answered Mike.

"Is there anything you can do to help complete any of the remaining Sprint work?" Cassie thought she could almost see the wheels turning in Mike's head.

"I don't know," stammered Mike.

"Relax, Mike," giggled Cassie. "I'm not trying to put you on the spot. As a team member, you should always be looking at how you can help the team achieve the Sprint goals. That might mean that you jump in and do some testing or help write some code for a story that hasn't been completed yet."

"So instead of pulling in a lower-priority story, I help get the higher-priority story that is already in the Sprint done," said Mike.

Cassie smiled and said, "I couldn't have said it better myself."

"So you never can pull a story into a Sprint once sprint planning is over?" asked Kelly.

"I prefer to never say never." Cassie responded. "Always do what makes sense. There may be a situation where the best thing to do is to pull a story into the current Sprint, but it doesn't look to me like it's a good idea right now."

On the surface, it looks like a good idea. You are done with your Sprint work and want to pull a story from the next iteration into this one. You won't be able to finish the story, but story points are counted in the Sprint in which the story is accepted. It's like getting a head start on the next Sprint. Great idea, right?

Usually, no.

Is there anything that can be done to help the team achieve the Sprint goals? In other words, can you help swarm the highest-priority stories that remain? Can you work on some technical debt, or possibly refactor some code?

A story can be pulled into the current iteration if the PO and team agree that it's a good idea. The PO could possibly pull in something small that could be completed in the remaining Sprint time, or even set up a spike story to do a bit of research into something that is coming up.

It goes without saying that the stories being considered need to have been refined and tasked. Pulling additional work into an iteration should be at the bottom of the list of things to consider if a team member finishes his or her Sprint work early.

DAILY STAND-UPS OR SCRUM MEETINGS

The team was standing around the Scrum board, and looking quite listless. Cassie asked, "What's going on today, team? You guys look like you are at a funeral."

Kelly said, "I'll just come out and say it… I don't see the value of having this meeting every day."

Cassie rolled her eyes and said, "You realize that there is nobody looking over your shoulder anymore. Really, I'm serious here… You guys need to take ownership. Your manager is not the person who solves all of your problems and tells you what to work on anymore."

"You mean we are responsible for everything?" asked Jack.

"Richard is the functional manager here. His job is to provide the environment for the team to work in. He sustains what I like to call the factory."

"Is that why I heard you talking to him about something called factory issues the other day?" asked Kevin.

"Yes, Kevin. Factory issues are things that affect the team's ability to work—like desks or training. I was talking to him about making sure that the team had enough product licenses so that everybody on the team could compile code." This really wasn't what she wanted to talk about, but Cassie was happy to clarify the role of functional manager and the definition of factory issues. She continued, "This team needs to realize one fact. You are in charge. Take ownership of the work and manage yourselves."

Ganesh interrupted, "Hang on there, Cassie. Do you really mean that nobody is going to tell us what to do?"

Cassie took a deep breath, paused for a second to collect her thoughts, and replied. "The old Waterfall structure involved a managerial role that watched over development teams and made decisions about how the team worked. Managers assigned work and worried about how products were built. Now, the Scrum team members commit, or sign up for work. The team is responsible for delivering what they committed to delivering. The team decides how to deliver the functionality. Scrum teams must be self-directing and self-organized. The team makes its own choices and owns the outcome of these choices."

"So, it's up to us," said Jack.

Cassie smiled. "Yup. The PO will tell you what she wants, but it's up to you guys—the TEAM—to decide how to turn the PO's idea into working product. Every decision belongs to the team. From training requirements to what coding language to use. A Scrum team owns what it works on."

"That is a ton of responsibility," said Kelly. "How do we make this work?"

Cassie replied, "By talking to each other. That's why we have a daily stand-up meeting. It's not meant to be a meeting where you come and talk to me or to the Product Owner or your manager. You are supposed to talk to each other. Have conversations, even difficult ones. Reach resolutions. You are not a bunch of people thrown together…"

"We are a team," said Kevin.

'Yes, you guys are a team. I don't think you realize that fact yet. Some of you need to find your voice and speak up. No matter what you say, getting involved adds perspective and information to the conversation. It's unfair to expect Kevin or Jack to address everything. Yes, they are the most experienced team members; however, they aren't the only two people on the team. "Cassie leaned back against the wall and said, "That's why the daily stand-up, or daily Scrum meeting is so important. You guys need to come together as a team, talk about what has been accomplished so far, and make commitments to each other."

"What does that mean?" asked Sue.

"It means that you don't just say what you plan to do today. You commit to the team that this is what you plan to get done today. You may or may not be successful at what you say you plan to do, but the team now knows what is going on."

"So we can plan our work accordingly," said Jack.

"And offer to help if we can," added Ganesh.

They are starting to get it, Cassie thought to herself. She said, "That is why the stand-up meeting shouldn't last for an hour. It's a quick opportunity for the team to touch base, understand where we are, and what's planned for today. Like I said, nobody is going to tell you what to do. You guys need to figure it out."

The one thing that every team I have ever worked with has done consistently is complain about meetings. Specifically, the daily stand-up meeting. In the past, I've let teams get away with not having a stand-up meeting every day. We tried every other day, twice a week, even three times a week. Every time, the team failed in the same fashion.

Everything didn't get done. Stuff slipped through the cracks. The team didn't communicate, let alone collaborate. It became obvious that we needed to meet every day.

The issue is that the team didn't see the value of the daily stand-up meeting. I can't blame them. After all, they were used to the Waterfall status meeting. Team members would get way too detailed in their updates. As I put it—"in the weeds." Because of this, stand-up meetings would last longer than fifteen minutes. Quite a bit longer.

I started putting a large timer right next to the Scrum board. I intended to tell the team that I would only allow the meeting to last fifteen minutes, but I never had to say a word about the matter. The team started policing itself just because I had a large clock counting down in the front of the room.

The stand-up meeting is valuable. If the team doesn't see the value in a short, daily "touch base" meeting, then the Scrum Master needs to find out why. Keep the stand-up meeting within the proper time box and facilitate the meeting in such a way that discussion flows between team members. The daily stand-up meeting is not a status meeting. A status meeting is where you sit around a table with your manager and play a game called "cover your butt" with everybody else on your team. A stand-up meeting is a short meeting where the team makes commitments to each other—not to management.

SPRINT REVIEWS AND DEMOS

Annie was doing a great job running the review meeting. Several customers and stakeholders attended and seemed to be engaged in the process. As Annie demonstrated the functionality the team had provided, stakeholders asked questions and provided feedback. The whole team was also in the meeting, although some of them did not seem pleased with what was going on.

"I don't like it," whispered Jack.

"What don't you like?" asked Cassie.

"First, we prop this thing up with fake data. That's bad enough. Now we have to sit here and listen to customers tear it apart." Jack was trying to whisper, but was having a hard time.

Cassie responded, "Jack, we need to get feedback from our stakeholders to ensure that we are building the right thing."

"I get that," said Jack. "But it's hard to listen to this. You know how hard I worked on this feature."

Cassie bit her lip and said, "Jack, nobody is questioning how hard anybody worked on any feature. However, the team should welcome feedback that changes to what is being worked on. The greatest product in the world has no value if it doesn't do what our customers want."

Product Owners freak out about Sprint demos…

Because customers attend this particular meeting, there is a notion that everything needs to be slick and professional. While I can appreciate that, I find no need for slide decks and polished demos. I've heard it said that if you like sausage, you should never watch the sausage-making process. I disagree; I want to know what goes into what I'm eating. Same with the Sprint demo. We should be looking for feedback from our customers and stakeholders. In order to get honest feedback, the Scrum team needs to honestly present what was worked on during the Sprint. Sometimes that requires that the sausage-making process be visible. Instead of wasting time creating huge slide decks, show what has been built. I'm not saying to do away with slides completely. What I am saying is that the manifesto puts maximum value on working software. That's what the Product Owner should focus on during the Sprint review meeting—no matter how "unpolished" that may be.

Once the customers and stakeholders provide their feedback, the team needs to do something with it. The team needs to be flexible enough to radically change what they are doing and even be willing to throw away work if need be.

THE SPRINT RETROSPECTIVE

Cassie had just taken the team through a Sprint Retrospective meeting. At the end of the thirty-minute time box, everybody agreed on three items to work on. The idea was that they would take these action items and work on them during the next Sprint.

"Great meeting, team," said Cassie. "Let's take these three items and get them in the Backlog."

I wish I could play the sound of a needle being yanked across a record because it would fit in here. The looks on the faces of the team reflected both shock and bewilderment.

"The Backlog?" asked Kevin. "I thought you wanted us to keep the Backlog, as you put it, neat and tidy."

"I don't want a bunch of stuff we don't intend to ever do in the Backlog, Kevin," said Cassie. "I do, however want everything we plan on doing there. Putting the team retrospective action items into the Sprint Backlog ensures that we will address them before the next Retrospective meeting."

My friend and fellow Agile coach Jay Cohen loves to say, "If it's not in the Backlog, it doesn't get done."

It's a simple concept. Putting action items into the Sprint Backlog gives visibility to how the team plans on inspecting and adapting. This way, nothing is forgotten or swept under the rug.

Inspect-and-adapt is something I've been unknowingly doing all of my life. From the time I first touched a barbell, I've been working to get better. If you happen to walk into a hard-core powerlifting gym, you will notice that when somebody is lifting, everybody stops what they are doing. Not necessarily to watch the lifter, but to give verbal queues and coach.

"Head up!"

"Sit back!"

"Drive with the hips!"

Everybody gathers to support the person lifting by looking for flaws in their technique and weak points that need to be addressed. The human body works as one piece. That means all kinds of muscle groups need to work together to, for example, perform a heavy squat.

I've competed in all kinds of crazy strength sports during my years on planet Earth. If you set up some kind of contest where you had to move something heavy—I'd show up—and usually win…

In order to compete at this level, I had to understand what my strengths and weaknesses were. That's all well and good, but the bottom line was to try to turn those weaknesses into strengths while making sure that you mastered the stuff you were already good at.

Here's the rub: In order to do that, you had to do the stuff you didn't want to do. As I put it, "master the stuff you suck at."

For example, for years, I had a weak grip. It really hampered my deadlift. Every time I went heavy, the bar would pop out of my hands. After going through about a million excuses in my head, I faced the situation and focused on my weakness. It wasn't pleasant, but now I can bend frying pans with those hands. I can't remember the last time I dropped a deadlift.

Understand what you are good at, and don't be afraid to leave your comfort zone to improve on your weaknesses. Take the time to inspect and adapt. When you work on those weaknesses, make them visible. Let everybody know that this is what is being addressed and chart your progress. As I like to say, turn your weakness into strengths.

TECHNICAL SKILLS OF THE SCRUM MASTER

"Duane, this is just weird." Cassie was discussing a discovery she had made with Duane during their weekly meeting. "It seems that the less I know about the work going on, the better Scrum Master I am."

"Funny how that works," replied Duane. "I know you know your way around software development."

"That's right," said Cassie. "I worked as a developer for five years before I pursued the Scrum Master position."

"But now," Duane responded, "you are working with a team that uses a programming language you are not familiar with."

Cassie said, "That's right. I don't have a clue how they write this stuff."

Duane smiled and said, "So you can focus on the agility of the team. Your primary concern is making them better at Scrum."

The first team that I served as a Scrum Master worked on a product that I was very familiar with. I had worked on the product, knew the code, and was well versed in how to install, configure, and use it. You would think that all of the intimate knowledge that I had about this product would enable me to be an effective Scrum Master for the team that was working on it.

That was not the case. In fact, in hindsight, I would say that my knowledge of the product hindered my ability to be a servant leader to my team. I simply was "sucked in" to the daily ins and outs of software development and team activity. Instead of focusing on helping the team become more Agile, I focused on the development. I started acting like the dreaded development manager. Even worse, I was dangerously close to jumping in and helping the team. That is not what a Scrum Master should be doing.

This fact became crystal clear to me when I started serving a team that worked on a product with which I was unfamiliar. I found it easier to focus on overall team agility when there was no temptation to get involved in the technical work. A Scrum Master should understand the nuances of how software is sourced and built, but their primary role is being a servant-leader to the team.

BUILDING ENGINEERING INTO THE PRODUCT BACKLOG

"So how do you expect us to build anything that works if you don't let us plan things out up front?" Kevin was a bit perplexed. He was struggling with how to design what the team was working on. He discussed it with Cassie, and Cassie thought it was best they both talk to Duane about the subject.

"Kevin, in Agile there is no big design up front. The architecture is refined as the product evolves," Duane said.

Kevin looked at Duane as if he has three heads. "Come again?"

Duane said, "Let's start with defining what architecture is. It can be high-level design and proof of concept, or guidance on the best technology. It can also be viewed as the mapping of business value into technical requirements or developing use cases. To me, architecture is the foundation on which the organization is built. You can't expect a quality outcome if your code is not built and organized for quality. Would you agree with that?"

"Oh yes," said Kevin. "Poor design can mess you up for as long time. If you pick the wrong technology, your code is hard to integrate. You almost need to refactor everything."

"That's why Scrum changes the way we look at engineering," said Duane. "We don't start with doing a ton of research and generating an architecture document, which was huge. As I'm sure you are well aware, we are now working with epics and user stories which we strive to keep as small as possible. There is no big design up front. As I said, Scrum teams refine architecture as products evolve."

Cassie chimed in: "That's why we have an architecture team here. Their job is to work with Scrum teams and to influence how architecture evolves. Every Scrum team

has a tech lead, or architect they should be working with, or preferably part of the architecture team to get architecture defined in the Backlog."

"So, these stories are prioritized?" asked Kevin.

"Yes," replied Duane. "They are treated like any other Backlog item."

"OK, but how do I get going?" asked Kevin.

Duane smiled and sat back in his chair. "Kevin, have you ever heard of a spike?"

"Can't say that I have," answered Kevin.

"A spike is a small story you use to go off and do research. I usually insist that a spike not take up a whole Sprint. The result of a spike should be enough information to write stories."

"Do you mean one of those architecture stories you were talking about?" asked Kevin.

"Maybe," answered Duane. "Or regular user stories that build value."

Just because we are doing Scrum doesn't mean that there is no architecture. Scrum relies on what is called emergent architecture. Emergent architecture emerges (hence the name) as work is done, and comes from within the Scrum team. It is usually discovered during story refinement.

Having an engineering team that works with the Scrum teams to create vision ensures that Scrum teams are looking at the big picture. Cooperation between developers, testers, architects, and others helps to create and capture new ideas. I've heard it said that architecture is a team sport. I couldn't agree more. Looking at how what is progressing in the current Sprint affects the overall architecture. This is a good thing. The architect should help the team identify what architecture is necessary and help them to implement it. This is a bottom-up approach that allows the teams to determine the specifics of design and architecture.

This chapter detailed the Scrum Master's responsibilities and core functions in the third person through the eyes of a fictitious Scrum Master working with a new Scrum team. In the next chapter, we will explore the Scrum Master's interactions with other roles.

CHAPTER 7

The Scrum Master's Interaction with Other Roles

To me, a Scrum Master is more about soft skills than technical proficiency or being super-organized. Here is my take on how a Scrum Master should interact with a variety of roles.

The Product Owner

"What you need to do is go dark for a couple of days."

I was working with a new Product Owner who was trying to put a release plan together, but was getting pulled in a bunch of different directions. Every time he would carve out some time to work on the release plan, something or somebody else needed his time.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_7

My solution to this dilemma was to "go dark." We went into a conference room and closed the door. He didn't look at his email, turned off his messaging software, and worked on the release plan with me. I actually told him to tell everybody that he was sick, but he didn't go for that. In retrospect, that wasn't being honest and transparent, so I'm glad he didn't. We focused on getting the release plan done and were successful.

That's a bit of an extreme example. I usually coach the Product Owner to be available to the development team as much as possible, but this was a special case. The PO has a lot of responsibilities that require a lot of support from the Scrum Master.

The PO has a tough job. I call them the one wringable neck. The one person who is responsible to both the stakeholders and the team. The Scrum Master needs to support the PO and help them create great products with the right features.

Supporting the PO is one of the best ways that the Scrum Master can lead the team. I like to say that the Scrum Master is the Product Owner's sidekick.

Be Available to the Team

The Scrum Master should coach the Product Owner to be as available as possible to the team. The PO should be doing whatever is possible to get the customers engaged with the team, but not to the detriment of the team. If the PO is only around for Sprint planning and the Sprint demo, they are doing the team a disservice. The team needs to be able to ask the PO questions while they are building what has been asked for.

When I teach Scrum, I like to run simulations. They give the students an opportunity to use what they have learned, and they get everybody up and moving. I like to have the students build things with little plastic blocks that snap together. I find that having the class do something that does not require coding or technology enables anybody to take part regardless of background. It's also hard to get folks who work in technology not to think about technology. For example, I used to run a simulation that asked the class to build a hypothetical website for a pizza shop. There was no coding involved, but the students would argue what type of web server to use or how to best code the JavaScript. That doesn't happen with plastic blocks.

What does happen is that the team gets so engrossed in the work that they forget about the PO. They are building a two-story structure and don't ask about details like how many windows, or where the door should be or if the structure needs a bathroom.

Same thing with a Scrum team working in an iteration.

Acting as Product Owner when I reject their work, it's a great opportunity to remind the students that building something that the customer doesn't want is a waste of time, and the PO is the voice of the customer. Ensure that the PO is available as possible to the team and that the team is interacting with the PO.

Build the Backlog

The most important thing that the Product Owner does is to build the Backlog. As I said before, the PO does not write all of the stories in the Backlog. The PO is, however, responsible for each story in the Backlog. Product owners should reprioritize the Backlog often. I like to see it happen daily. Yes, it might be overkill, but I like the POs to be thinking about the stories in the Backlog. Reprioritizing the Backlog daily guarantees that the PO is very familiar with the stories in the Backlog. That's what is really important. The PO needs to be intimately familiar with the stories in the Backlog so he or she can effectively articulate the customer value contained within. The Scrum Master needs to ensure that the PO is able to focus his or her energies on the Backlog.

The Scrum Master should coach the Product Owner to create purpose with the user stories—just like Cassie coached Annie in Chapter 6. Too often, I see stories that emphasize how the team is to build something. The PO should focus on letting the team know what the user wants and, more importantly, why they want it. The "why" is critical. The team needs to view user stories through the lens of the customer. By making it crystal clear why the customer wants this thing, the PO helps the team focus on delivering that value. As a Scrum Master, I've been known to sit in Refinement meetings and continuously ask why the customer wants this thing.

Help Out

As a Scrum Master, I'm looking to help the Product Owner out as much as possible. I'll jump in and do whatever I can do to make the PO's job easier. I won't actually write stories, but one way the Scrum Master can help out is by facilitating meetings. For example, facilitate the Sprint Review and demo. The PO is there, and does a lot of the presentation. Yes, the development team does the demo, but the PO is the person with whom the customers and stakeholders have been interacting. It's natural for the PO to serve as a sort of master of ceremonies for the meeting. The Scrum Master can set the meeting up, organize the stakeholders, and help run the presentation. The PO is still the focal point, but the Scrum Master is taking responsibility for facilitation.

The same thing can be done with the Refinement meeting. Facilitate whenever you can. Take the responsibility of running the meeting from the Product Owner so that they can focus on the stories with the team. Story refinement needs to happen at least once a week. Too often I see the first thing to be

removed when things get hectic is the Refinement meeting. Work with the PO to ensure that the team is refining stories weekly.

The Scrum Master needs to ensure that the PO is empowered to do their job. The PO needs to be able to make decisions quickly. If he or she needs to go to management with every decision, the team is not going to be successful. Often, Agility stops at the team level. Scrum requires that the entire organization operate in an Agile fashion. Management needs to give the PO the authority to make decisions and the ability to deal with their consequences. The Scrum Master can make a real difference here. Work with management to ensure that they understand the importance of the PO role and coach them to trust that person to turn customer requirements into user stories that deliver value.

I always make it a point to check in with the Product Owner often—daily, if possible. This way, I find out what is going on inside their heads and do whatever is necessary to help them mature as a PO. Once, I had a PO tell me it would be great if they could visualize the whole Backlog. I wrote out each story in the Backlog on sticky notes and covered a conference room wall with them. I'll admit that it was a bit of overkill, but the PO was able to visualize the Backlog.

The PO will always push for more. It's the nature of the job; after all, they are the one wringable neck. The Scrum Master should be coaching the PO to allow the team to make commitments and deliver on them. Overloading the team isn't going to do anything but create problems. The PO can push, but the Scrum Master should push back. Remind the PO that he or she should be focusing on throughput. The team should be working at a sustainable pace, and quality is never sacrificed to meet any type of schedule. The PO should allow the team to get work done while he or she takes the time to be sure that stories are ready for them. Once the team gets used to not overcommitting, and to getting stories done, they will pick up velocity.

The Dev Team

Human beings are made to be in community. In fact, we crave it. We gravitate toward like-minded individuals. I've heard it said that a Scrum team is a tribe. That's a great analogy, but I prefer to think of the Scrum team as a band.

Scrum as a Garage Band

A band is trying to achieve harmony. In order to do this, each musician must know their role and play their part. The guitar is usually the focal point of the band. A bass guitar is more than a guitar with two fewer strings. The bass player takes care of the low end of the musical spectrum and is the foundation to the wall of sound a band creates. Usually the bass plays an octave lower

than the guitar. Playing the same notes as the guitar doesn't sound right. The bass and drums lay the foundation of the music and drive the rhythm. When I play bass guitar, I try to keep the beat with the drummer's bass and snare drums. I want to make the drums and bass coexist so the guitars and keys can reach soaring highs, with the drums and bass providing the pulse behind them. When done correctly, it sounds great. When everybody does not stay within their role, it sounds like a mess.

The Scrum Master is responsible for keeping the team in rhythm and on key. Just like a musical group, each member of a Scrum team should know how to play their part and create harmony.

There is a reason I picked a band and not an orchestra. Musicians in an orchestra play the music exactly as it is written. Folks in a band noodle around with notes and chords and figure out how everything fits together. Playing is more about collaboration than following the music exactly.

Scrum Harmony

So how can a Scrum Master help a team achieve harmony? First, understand that transformation is a process. Understanding where the Scrum team is in the process will help the Scrum Master coach them and improve. For example, let's say you want to learn how to make a killer grilled cheese sandwich. Here are the levels that you will pass through:

- **Awareness**: You have tried a grilled cheese sandwich and liked it. You decide that you might want to learn how to make one yourself.
- **Competency**: You can make a sandwich and not burn it most of the time. In the end, your sandwich is edible.
- **Proficiency**: You can turn out grilled cheese sandwiches like nobody's business, and they are really, really good.
- **Complexity**: You start to experiment with different types of cheeses and breads and go from making white bread and American cheese sandwiches to more of a gourmet offering. You may start to help and mentor other folks in the fine art of grilled cheese.
- **Mastery**: You are the "go to" source for all things grilled cheese.

Why am I talking about grilled cheese? Admittedly, I'm food-driven, but it is a simple example of the stages a team will go through when learning a new discipline. I'm not talking about making dashboards and reports look good.

I want to deliver great software that our customers can't wait to get their hands on.

The Scrum Master should be constantly coaching the team on how to get better at Scrum. It's important that the Scrum Master help the team to take an honest look at themselves and acknowledge where they are in this process.

So my, ahem, cheesy list looks like this when I apply it to Scrum:

- **Awareness**: You have an understanding of what Scrum is, and you may have discussed it with some folks.
- **Competency**: You are now "doing Agile"—which in reality means that you are having a lot of meetings and fumbling around with trying to make this Scrum thing work.
- **Proficiency**: You actually are getting good at this. You deliver features each Sprint, and your burn-down isn't terrible. As an extra bonus, there is no blood on the meeting room walls after Sprint Planning.
- **Complexity**: You begin to modify things to make Scrum work better for you and your team. You are consistently demonstrating Agile techniques and are identifying where to experiment with new practices. You are no longer "doing Agile." You are "living Agile."
- **Mastery**: The team is producing value at a sustainable pace while adapting to the changing needs of your customers.

Knowing where the team is along this path helps the Scrum Master understand how to best coach the team. If the team is in the competency stage, they need to not modify anything in the Scrum framework. In other words, the guard rails need to be kind of rigid. Remember, things are going to feel weird, and the team is going to want to go back to their familiar Waterfall practices. As the team matures, they will begin to see the value of Scrum and embrace the value of the ceremonies. As the team inspects and adapts, the guard rails can get softer because the team won't be running back to their formerly comfortable Waterfall ways. The team will want to experiment and figure out the best way to become more Agile and achieve mastery.

Leader, Fan, and Cheerleader

The Scrum Master is the servant-leader of the team. In reality, the Scrum Master has no authority. In other words, the Scrum Master can't make the team do anything. That doesn't mean that the Scrum Master can't be effective. The secret is that the Scrum Master needs to convince the team that he or she is "all in"—that the only reason they come to work every day is to make the team a success.

I like to say that I'm the team's number one fan and cheerleader. As a Scrum Master, I spend a lot of my time encouraging team members, watching, and listening. I like to observe the team dynamic and tweak things to promote collaboration and teamwork. For example, if a team member isn't taking part in the meeting, I'll try to direct the conversation to them. If somebody on the team is talking about some uber-technical subject, I might ask him or her to clarify, even if I have no idea what they are talking about. I'm also constantly gauging the value of team conversations. Is this something that should be in a parking lot, or is it valuable to let it go? I also believe in being the "C and C team builder." C and C stands for congratulate and celebrate. When is the last time somebody said "thank you" and caught you off guard? I'm not talking about when a bank teller thanks you after he or she completes your transaction. I mean when somebody sincerely wants to thank you for something you've done. How did that make you feel?

Human beings need to feel appreciated. Appreciation confirms that the work you have done is valued. Knowing this, I'm always looking for ways to show appreciation. When the team accomplishes something, even when it's something they should be doing anyway, I congratulate them for a job well done. For example, if the team finishes all of the stories they committed to in a Sprint, I'll make it a point to congratulate them for a job well done. There is power in in appreciation, and it happens way too infrequently in today's business culture.

Celebrate whenever possible, and not just for the "normal" reasons like birthdays. Don't get me wrong here. As I said before, the Scrum Master should know every team member's birthday and hold team birthday parties. I do think that is important because it shows that we care about the human being. After all, resources don't have birthdays.

Take it to the next level. Celebrate team success. In the example, after congratulating the team for finishing all of the stories in the Sprint, take them out for coffee. By coffee, I don't mean the stuff in the break room. Go out for an hour or so and enjoy some time with the team. Obviously, get the functional manager's buy-in before suggesting it to the team. That should not be an issue, as the functional manager should be interested in building a good team dynamic as well.

I can remember talking to a team member who casually mentioned that he and his wife were expecting their first child. "So when is your wife due?" I asked.

"Next week," he replied.

I was shocked that I hadn't heard about this before so I checked around and found that he didn't tell anybody on the team about this event. Admittedly he was an "all business" kind of person, but I still was amazed that he hadn't shared a significant life event with any of us. I brought the team together (without the team member having a child) and suggested that we take up a

collection and get the new parents and child some gifts. The team was not only agreeable, they were excited to do it.

After the child was born, we presented him with the gifts we had purchased. The new father was touched that the team would do such a thing. I know the impact of something like this. Back when my daughter Jorden was born, the team I was on did the same thing. I know how it made me feel, and I want others to feel as good as I did that day.

Doing stuff like this builds the bond between team members. Did you ever notice that the offensive line of a football team always sits together when they aren't on the field? It's not usually because the coach makes them sit together. Don't get me wrong. He wants them together so he can find them when he needs to talk to them, but they don't need to be told. They do it naturally, usually by position. They are a team within a team, and the bond between them is a strong one. As a Scrum Master, I'm trying to build camaraderie and chemistry between team members.

I'm a big believer in stopping in on each team member daily. I just want to see how things are going, and if there is any way I can help out. As a Scrum Master I want to build rapport with every team member. I want them to trust me and believe that I'm looking out for them.

I want to build up the team in every way possible. I want the team to be in sync (harmony again). I'd like to get to the point where they know each other's tendencies and rhythm so they know how to work with each other in a way that leads to high performance.

Work-Life Balance

As the story goes, nobody ever said lying on their deathbed that they wished they could have spent more time at work. As I've said before, Scrum is different. Teams commit to the work, and then deliver. As a Scrum Master, make sure that the work that team members pick will not be too much for them. I remind teams all the time that the only thing you receive from working long hours and weekends is burnout. It's important to coach the team to keep that balance between what is good for the person, good for the company, and good for the team.

Other Scrum Masters

How do you get better at being a Scrum Master?

I've been asked more than once what to do to become a better Scrum Master. I usually answer as you would expect. Study up on the craft. Read books and

blogs. Attend conferences and seminars. Enter a coaching relationship and, most importantly, become a member of a Scrum Master community of some type. Get around like-minded individuals so you can learn from them, and they can learn from you. The Scrum Master role can be demanding. It's helpful to have a support network of other Scrum Masters to lean on.

Community

Get together with other Scrum Masters and talk about what is happening in your teams. Different perspectives provide insight into the situation that may not be obvious to the person immersed in it. Like the old saying goes—two heads are better than one. Something else I stress quite often is that what happens in the Scrum Master meeting stays in the Scrum Master meeting. Keep the conversations private between the Scrum Masters. Don't let them leave the room. You should be able to open up and be honest with each other. Talk about each other's struggles. Give opinions and share solutions. By doing this, you will gain insight into how to better serve your team.

For example, there are probably hundreds of ways to do a Retrospective meeting. I like to talk about what techniques worked well and which ones didn't go so well. You can also talk about what happened with each of the Scrum teams, but be careful. Do not share anything that the team doesn't want to be shared. Respect the team boundaries, but share.

One of the things that always amazed me about powerlifting was the way the lifters treated each other. You would think that a bunch of folks who were competing against each other would be eyeing each other up like a hungry wolf looking at a pot roast. Not so. Everybody was there to compete, but we all looked out for each other. Powerlifters need a lot of help on days when they compete. They need help with equipment, such as tightening those heavy belts, or wrapping knees before squat attempts, or help getting into super-tight supportive suits.

Support each other. When a Scrum Master is out sick or vacationing on a beach somewhere, step in and facilitate meetings for them. Maybe even buy the donuts for somebody else's Scrum team every once in a while.

The Agile Coach

Everybody has a comfort zone. It's a, well, *comfortable* place where there is little stress or risk. It's not a bad place. Most folks can perform consistently while in their comfort zone; however, you will never achieve high performance if you don't step out of your comfort zone. Part of what an Agile coach does is to encourage you to stretch yourself and get you out of your comfort zone. Maybe a better way to say that is the coach can push you out of your comfort zone.

Get Out of Your Comfort Zone

When I was coaching girls' basketball, I would purposely make the girls dribble twice as much with their nondominant hand. I was trying to make them harder to defend. What happens in basketball (and with Scrum teams) is that when under pressure, players go back to what they trust. They fall back into their comfort zone. If you only trust dribbling with, say, your right hand, that's what you will fall back to. If I'm trying to defend you, I'll know that I can look for the ball to be on your right side when I apply pressure. That makes the ball easier to steal, especially if you are trying to move left. Eventually, some of them got so good with their nondominant hand that they trusted their ability in both hands.

A good coach can keep a Scrum Master accountable while encouraging them to try new things and learn from failure. The Scrum Master is focused on the team. The Agile coach is focused on the bigger picture. The coach is not a member of any Scrum team, and is focused on the organization. He or she supports the Scrum Masters and the teams they are serving. The coach is usually much more experienced than the Scrum Masters and can apply knowledge from a variety of methods such as Scrum, Lean, and Kanban. A coach primarily focuses on Scrum Masters, but can coach and mentor anybody—from teams to management to executives.

A good coach is like yeast. They spread to everybody and help them grow. A Scrum Master's relationship to a coach should be exactly what it sounds like. Think of a Sensei and student. The coach is there to help the Scrum Master grow and learn.

Over the past couple of years I've spent my Saturday mornings with Tony Pharr, shown in Figure 7-1. We were both in the gym again, but we weren't getting ready for competition. Well, we weren't getting ourselves ready for competition. We were working with the next generation. We were coaching some of Tony's younger relatives and my nephew Max. They were in the weight room getting ready for the upcoming football season. Tony was up to old tricks, telling them they had to lose to win while gauging their effort as they strained against the bar. I was changing weights and calling squat depth while adding my own encouragement. We definitely pushed them out of their comfort zone, and they became strong young men as a result.

Figure 7-1. Tony Pharr, world champion powerlifter

Don't underestimate the power of coaching. Both the Agile coach and Scrum Master will coach as part of their day-to-day activities, and coaching is possibly the most important thing either one of them will do.

Jake Turner was my best friend's uncle. The guy who took me and my friends under his wing and taught us all about powerlifting.

I still talk to Jake Turner on occasion. Now in his early seventies, he no longer lifts weights, but still has the quick wit and twinkle in his eye I remember from way back when he took a bunch of snot-nosed kids into his garage and taught them how to lift weights. He was amazed that I was going to write about him in this book. He didn't think he did anything special for me.

Boy, was he wrong.

If he realized it or not, what Jake did for me is a great example of what a coach does. A coach sees value in everybody and helps to bring that value out. At a time in my life when I didn't think much of myself, he coached greatness out of

me. No, I didn't go to the Olympics or win a world championship in powerlifting. I did, however, realize that I could do anything if I put my mind to it. Now I do the same things with Scrum teams.

Managers

The Scrum Master and manager should be on the same page. The roles are similar. Both are looking to give the team what it needs to succeed. The big difference is that the Scrum Master has no authority. The manager does.

The Scrum Master should share almost everything with the manager. The only thing that I do not share with the team manager is stuff from the Retrospective that the team does not want to leave the room. It's the same thing with information shared in a coaching session that is agreed not to leave the room. Everything else is shared. I want to give the manager my perspective on the inner workings of the team.

Managers sometimes struggle with their role in Agile. They no longer tell folks what to do. The team now decides how they are going to go about their business. The manager role is still valuable. It's just not steeped in command-and-control anymore. The team does not need to be micro-managed.

The manager's role in Scrum is to nurture the professional development of team members. The manager holds regular one-on-one meetings with team members. The manager provides mentoring and coaching in these meetings. The manager also takes care of the team financials. They are also responsible for performance reviews.

The manager can also assist the Scrum Master with removing impediments. Since they do have authority, the manager can really help the Scrum Master in this area.

The Scrum Master's Interactions with Stakeholder Roles

I've mentioned stakeholders more than once to this point in this book. I'd usually talk about users and stakeholders, but I never identified who the stakeholders were. This is by no means a comprehensive list. I'm going to detail the roles that I'm most familiar with.

Project Managers

The Project Manager provides end-to-end support for the project. Think about Scrum at the team level: the team produces value, the Scrum Master protects the team and enhances the Agile process, and the PO interacts with customers

and build the Backlog. Who takes care of the other stuff like making sure we are in line with corporate governance so we can release? The project manager. The project manager keeps track of things like making sure support is properly trained, that all of the legal requirements needed to release the software are met, that the project is on time, and probably a whole pile of other stuff I'm not aware of. The Scrum Master needs to keep the project manager in the loop about what's going on with the team. Usually the Scrum Master or Product Owner will share information like the release burn-up and team velocity with the project manager to help give an accurate picture if the release is within scope or not. I like to think of it this way: The Scrum Master reports down to the team about the health of the project, and the project manager reports up through the business.

The Sponsor

Sometimes known as business sponsors, project sponsors provide exposure and sometimes motivation for the team at a high level. Normally, they control the purse strings of the project, so they are the person who is very interested in things like return on investment (ROI). In reality, everybody on the team is accountable to the sponsor. The Scrum Master really doesn't interact with the sponsor, or sponsors, very much. If an impediment is beyond the team's span of control, the sponsor can get involved, as they should have the authority to help. The project manager usually is the one who interacts with the sponsor.

Support

The folks who get involved when customers have problems are an untapped resource. I used to be a support rep a long time ago. So long ago that I used to define my role as "the guy who answers the phone when you are in trouble." Support reps don't use the phone so much anymore, but they do interact with customers. The right kind of customers. The ones I refer to as "on the glass." The folks who actually use what the Scrum teams produce.

The Scrum Master should include support people in as many Scrum ceremonies as possible. They can attend daily stand-up meetings to get an idea of what the team is building. They should be at the Sprint demo so that they can see the functionality in action. They should be talking to the POs about the kind of customer issues they are seeing so that the PO has an idea of some of the difficulties being experienced in the field. This isn't just about defects, but the customer experience as a whole. The software can be defect-free, but if the install process is an ordeal, the customer probably isn't happy. The PO can recognize this and create Backlog stories to address it.

The Scrum Master should talk to the support reps regularly to ensure that they are getting the training they need on the new software being produced.

Often, support is the "face" of the company. Make sure they have what they need to make the customer happy.

Users

The Product Owner doesn't have exclusive access to the user. I coach the teams that I work with not to be afraid to talk to users at any time during an iteration. We want users to be part of the development process, so it makes sense that they should be able to talk to anybody on the team when they are available.

That includes the Scrum Master. Tell yourself, "Look, I'm not going to be able to talk about technical details or product features, but I am there to support the team. If I can help a customer out, I will."

In this chapter, we looked at how the Scrum Master interacts with other Scrum roles. The Scrum Master is the servant-leader of the team first, and an Agile champion to everybody else. In the next chapter, we will explore the soft skills that a Scrum Master needs.

PART

III

The Scrum Master's Skill Sets

CHAPTER

8

Soft Skills of the Scrum Master

A Scrum Master is the servant-leader of the Scrum team. That's easy to say, but what does it really mean?

To me, a Scrum Master needs to be a coach and a constant enabler of change. It's up to the Scrum Master to keep the Agile flame burning bright. To constantly remind the team of what they are trying to achieve, and coach them so they actually get there.

My brother Jason recently switched careers. He'd been an elementary school teacher and football coach for as long as I can remember. Imagine my surprise when I got a phone call one day where he asked me what a Product Owner was, because he'd joined a software company. Product Owner was his new role.

Yeah, he's been talking to his big brother a lot since then…

One of the things that became obvious to Jason very quickly is that there is a difference between professional coaching and what you see in sports.

The Professional Coach

What do you think of when I say the word "coach?" Usually, the images that come to mind are chairs being thrown across a basketball court, or the red-faced football coach berating a referee after a call didn't go his way.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_8

I've experienced good coaching and what I would classify as terrible coaching in my life. I once had a football coach tell me to "suck it up" as I lay on the field with a broken ankle. That wasn't the end of it. At the end of practice, the team left the field. I couldn't. The practice field was located below the field. To get back to the locker room, you had to go up a set of steps and then walk up a hill. With my injury, I couldn't navigate the steps. I was stuck on the practice field in all my gear. My coach left me there.

Eventually, a couple of my friends came looking for me. They helped me hobble into the locker room so that I could change out of my equipment. As you would probably guess, the lack of concern and empathy in this situation did not have a positive impact on my early teenage psyche. That coach, whose name I can't remember, made an impact on me.

I've also had positive coaching experiences as well. Jake Turner, whom I consider my first powerlifting coach, used to say that my squat form resembled a monkey doing something with a football that I can't put in this book. Let's just say it was bad… really bad. In powerlifting, the squat is one of the three lifts used to demonstrate strength. To perform a squat, you place a loaded barbell across your back and do a deep knee bend. The lift is judged on depth. The crease of your hip must be below the top of the kneecap when viewed from the side.

At first, I couldn't hit depth with a broomstick across my back, let alone a loaded barbell. Instead of writing me off as a lost cause, he coached me. He used to say that my legs were plenty strong, I just didn't know how to use them. He worked with me, gave me the proper verbal queues, and built my self-confidence. Jake poured confidence and knowledge into me, and the result was a pretty good powerlifter, if I do say so myself. One of my proudest moments was having Jake see me as the last lifter to squat in a powerlifting contest. That meant I was attempting the heaviest squat of the day. There were no monkeys or footballs around that day primarily because he took the time to coach me.

Coaching is not about blowing a whistle and making people run laps when they mess up. To me, coaching is about bringing greatness out of people. In order to be an effective coach, remember one thing: Everybody is valuable.

That's it. Every person you come in contact with during the day has something to give. Your job as coach is to bring it out of them.

Getting the Team to Buy In

I'll be honest, most teams don't embrace Agile at first. It's up to the Scrum Master to get them to understand and realize the benefits of Agile. I like to use a technique called beaconing. To illustrate what I'm talking about, I'd like to use the biblical story of Joshua. As the story goes, when the twelve tribes of

Israel crossed the Jordan river into the promised land, Joshua had the leader of each tribe take a rock from the river and add it to the stack of rocks on the opposite bank. When everybody had crossed the river, there where twelve rocks stacked up. When asked why he asked the tribes to do this act, Joshua said that these rocks were a monument to the great thing that had happened on this day. In the future, when your children ask why that pile of rocks is there, you can tell them the story of the twelve tribes crossing the Jordan river.

I try to do the same thing with Scrum teams. No, I don't make them stack rocks. I continually remind them of the good things that have happened. When the team has a good experience with Agile, that's a beacon. Let's say the team slightly changed what was being built in reaction to stakeholder feedback. I would remind the team of that behavior shift every chance I got. When things were going well, I would point to the beacon. When things were not going as well as expected, I'd point to the beacon. I continually and purposely reminded the team that they are becoming more Agile. Constantly reminding the team that they can be successful at Scrum causes the culture to shift. Small wins build trust. Let those small wins start to pile up, and the team will start to trust the process. Work will start to flow. Then the team will get to the point where it is able to make more decisions, faster.

Communicating Up Front and Often

Transparency is one of the pillars of Scrum. Sometimes, being transparent isn't as easy as it sounds. For example, it is extremely hard to get a team to be totally open when things are not going well. This requires trust, and a lot of Scrum teams have trust issues from previous experiences with command-and-control.

Think about this. Five really big offensive linemen protect a much smaller quarterback. The linemen need to give the quarterback time to throw the ball, move the team down the field, and score. The quarterback has to trust that the linemen will do their jobs, so he can focus on completing the pass. If there is no trust established, the quarterback's eyes will be on the blitzing defenders, not downfield. Bad things will happen.

Command–and-control comes down to a lack of trust—at least in my mind. When the team isn't trusted, micromanagement happens. The manager will become very interested in who does what on a day-to-day basis as opposed to how the team is delivering. There is an emotional side to trust as well. You know, where you expose your shortcomings in hopes that your openness will not be taken advantage of. Asking a team to operate in a transparent manner is not comfortable for a team, probably because of the command-and-control sins of the past.

Agile is really built on trust, when you think about it. Instead of a director or manager "managing" a software project, the Scrum team is given a great deal of freedom. It is the Scrum team that determines how the work will be done and when it will be complete. The Product Owner needs to trust the team in this respect. If the PO turns into a development manager, command and control begins to creep back into the team dynamic. This leads to team silos, low morale, and increased work pressure, and it will reduce the quality of the software.

A good Scrum Master needs to help the team get past trust issues and become more transparent. Communication is critical. The most obvious place for this to happen is the daily stand-up meeting. Every day, the team should walk out of that 15-minute meeting with a complete understanding of what is going on.

But what about people that aren't part of the team? Sure, anybody can attend the daily stand-up, but I coach my teams to be so transparent that anybody, from the CEO to the janitor, knows what is going on. I like to use *information radiators* to do this. Information about each Scrum team's collective progress should be on display. As I put it, "proudly, even obnoxiously displayed in a public place." A radiator should show Sprint information such as the Scrum board, Sprint burn-down chart, release burn-up chart, and any impediments that the Scrum Master is working on removing. Over-communicate all of the time, in every conversation you have with every person you come in contact with.

Listening Responsively as Servant Leader

As I said earlier, my wife says that I don't listen to her. I, of course, disagree. After all, I'm sitting right next to her…

But, she's right. I'm not listening to her. I'm recounting last night's baseball game in my head, or thinking up something witty to say, or maybe I'm just daydreaming. The bottom line is that I'm not focused on her. My attention is elsewhere.

Multitasking kills conversation. In order to truly hear what somebody is trying to say, you need to focus your full attention on that person. Obviously, that means put your phone down. When I'm talking to someone, I make it a point to put my phone face down on the table or desk. If there isn't anywhere to place my phone, I put it in my pocket. I do not want anything to distract me from the person I'm talking to, and that's what a smartphone does. If I'm checking my email, texting, or updating social media during a conversation, I'm not paying full attention to the person talking.

I'm not listening.

If my coaching mantra is true and everybody truly is valuable, then they deserve your attention. However, removing distractions and focusing on the conversation is not enough.

Good listening also requires you to turn your brain off and truly focus on what is being said. When I'm in a conversation, my brain is constantly working. It's just the way I am. I have to fight the temptation to go off and think of a solution, or what I am going to say when it's my turn to talk, or what I'm going to eat. When I'm trying to really focus on a person who is talking, the conversation in my head goes like this:

"I wonder if we need to…"

"Stop it! You're doing it again…"

"FOCUS!"

I need to constantly focus on the speaker and not let my mind wander off.

I also like to rephrase what the speaker is saying and reflect it back to them. At certain points in the conversation I'll say. "My understanding of what you said is…" and I'll paraphrase what the speaker just said. This gives the speaker the chance to verify my understanding of the conversation.

Use the Proper Tool

A good carpenter has a toolbox and knows which tool to use in a given situation. My father owned a construction company. When I was a teenager, I worked for my father in the summer. I'm here to tell you, construction work was not for me. I just didn't have the necessary skills to do the work. I am grateful to my father for those awkward summers I spent hitting my thumb with a hammer and falling off roofs. They showed me that I wasn't going to make it in the family business. Instead of construction, I got into information technology. I did pick up a few things during my time in the construction business. One was a saying my father had: "Tooling up!"

My dad would say that every morning before his crew went up to work on the roof. The idea is to have all the tools you would need for the day.

A coach has tools at his disposal. One tool is coaching. When coaching, the coach understands that the person being coached already knows the answer that they are looking for. It needs to be drawn out of them.

Another tool is mentoring. I like to describe mentoring as "been there, done that." When mentoring, the coach has experience with the type of situation on question, and explains how the problem was solved in the past.

Last, there is teaching. Teaching is where the coach transfers knowledge to the student.

Like a carpenter, a good coach knows when coaching, mentoring, or teaching fits a situation. Sometimes, the situation requires that all three techniques be used. Make sure that as a coach, you use the proper tool for the job.

Breaking Up Fights

I worked as what is commonly referred to as a bouncer to supplement my income back before I got married. The job is exactly as you would imagine. I spent most of my time at the front door checking everybody's identification before they entered the nightclub. I was to ensure that nobody who was underage was able to get in. When a fight broke out, I was referred to as "WMD" or the weapon of mass destruction. This was not because of my awesome fighting skills (which I do not possess). It was because of my size. When I showed up, the fight usually stopped. My job as a bouncer was not to beat people up. It was to get customers who were fighting, or about to fight, out of the club before they caused any damage or bodily harm.

I like to say that I've broken up more fights as a Scrum Master than I did as a bouncer. Maybe not fistfights, but folks in a Scrum team certainly experience a lot of strong disagreements. Sometimes people are having a bad day and act in a less than professional manner. Sometimes, old Waterfall behavior flares up—for example, disagreements between developers and testers. To put it bluntly, sometimes folks just freak out. I like to think that most people are not psychopaths, so such behavior usually has a root cause. People have families and pressures both at work and outside of work that may cause them to become overwhelmed. When somebody is having a bad day, there is usually a reason behind it. It's the same thing when two team members are at odds. Normally, there is a reason behind the disagreement. It's up to the Scrum Master and coach to help these people work through these issues.

A Scrum Master is a coach. Yes, there is an Agile Coach role, and that person's job is to coach both the Scrum Masters and Scrum teams. The Scrum Master, however, is embedded with the team. They will have a familiarity with the people and personalities on the Scrum team and should have built a level of trust that allows for a more impactful coaching experience than the Agile Coach can provide. A Scrum Master should be looking to coach anybody on the team whenever possible.

When conflicts arise, the first order of business is to limit any possible damage. Feelings can be easily hurt, but conflict can also be a healthy thing. In fact, a collaborative team should embrace healthy conflict. A Scrum Master should let conflict play out as long as it is healthy.

Let me clarify that thought. Let the conflict play out… to a point. Time-box the event so that the rest of the team isn't sitting in a meeting watching these two people go at it for an hour.

If conflict goes too long or moves from healthy to hurtful, step in and call for a parking lot. Stopping the event will diffuse any escalation and allow the participants to cool down a bit. At some point, the people having the conflict are going to have to sit down face to face and figure out how to work the

situation out. The Scrum Master needs to be very careful. If the parties involved aren't ready for a face-to-face meeting, schedule some one-on-one meetings with each individual—either with the Scrum Master or the manager.

Give people room to rant, but time-box this as well. I usually give a person ten minutes, then say something like, "Now that you have that off of your chest, what steps can we take toward a positive outcome?"

Guide everybody toward the positive steps that will solve the issue. Guide the team members toward a win-win situation. One of the responsibilities of the Scrum Master is to remove impediments. To me, internal team conflicts are a huge impediment that sometimes get overlooked. Problems between team members, or even the team and Product Owner, need to be brought to light and addressed. A team will never be high-performing with dysfunction within their ranks.

Conflict will result in either a resolution or an impasse. You want the parties in conflict to come to a consensus, which means everyone can live with it and support it going forward. The last thing you want is somebody muttering "No way" under their breath.

Handling Emotional, Overbearing, or Negative Behavior

How many times have you seen a professional athlete totally lose their mind? I'm talking about the football player who runs his mouth to the referee and has a 15-yard penalty assessed against him that ends up costing his team the game. Water coolers seem to get beat up quite a bit in baseball dugouts. At all levels of sport, frustration causes players and coaches to throw equipment, berate officials, or engage in a variety of less than professional-looking practices. There is no difference between a Scrum team and a sports team. A team is a team, and negative behavior is not what you want in that environment.

There is usually a reason for this type of behavior. Nine times out of ten, it's because of frustration. The Scrum Master needs to find out what is frustrating the team member and come up with a plan to address it. For example, let's say one of the senior developers on the team is acting out. The Scrum Master should meet with them over coffee and see if everything is alright. During the conversation, the team member shares that they are worried that they will become expendable if they give up their expertise. Nothing could be further from the truth. A Scrum team needs people with coding chops, and a senior developer achieved the position because of their ability to design and write code. What has changed is that we now value teamwork and collaboration.

Change is a scary thing, and it can frustrate people. So can team dynamics, management, and a myriad of other things. Dig into the root cause and take steps to address it.

Make it clear, however, that the team will not tolerate continued bad behavior. I've coached some teams that use yellow and red cards to help control behavior. These are based on the penalty cards issued by umpires and referees in the game of soccer. In soccer, a yellow card is a warning. For example, it is issued when a player is acting in an unsportsmanlike manner. When a player gets a yellow card, they can stay in the game. A red card is issued for flagrant fouls, and the player is ejected from the game.

We used yellow cards as a way to pause the meeting. It drew attention to the disruptive behavior of an individual, and usually led to a team discussion. A red flag was the nuclear option. If a red card was thrown, the meeting immediately stopped.

I'm happy to report I never experienced a red card being thrown.

Getting Through to Quiet People

Some people naturally shy away from the limelight. They are uncomfortable with speaking up. Either it's not their "thing," or there is some type of fear at play in this situation. Fear of rejection, fear of public speaking, maybe they are afraid to speak up because they lack self-confidence. No matter what the issue, a Scrum Master needs to facilitate the Agile ceremonies in a way that fosters communication for the entire team. Allowing only part of the team to dominate the conversation is doing a disservice to the team. To be a healthy, high-performing team, everybody needs to participate.

Some techniques can be used to ensure that the whole team is involved. A lot of Scrum team activities are designed to allow the whole team to participate—for example, the entire team throws their story point guess at the same time, or everybody puts a sticky note for each section of the retrospective. One thing the Scrum Master can do is to guide the conversation to the people who are not participating. Simply ask them to share their thoughts.

Another technique a Scrum Master can use is to give somebody who is trying to dominate the conversation a job. For example, ask them to write everybody's idea on a sticky note and then you take them and attach them to a flipchart.

Remember, everybody on the team is valuable. The Scrum Master needs to make each team member feel that they are a valuable member of the team.

Lightening the Mood

When I am asked to do a presentation, I like to start things off with a joke. It lightens the mood and makes people in the audience less defensive. It's the same thing with a Scrum team. A good Scrum Master knows how to read the room and when cracking a joke or doing something silly is just what the team needs.

I've worked for managers that would get upset if we allowed folks to change their desktop backgrounds. That's what I refer to as old data center management. Steeped in command-and-control, rigid managers viewed the people working for them as resources and were about as personable as a Tyrannosaurus Rex. They never would have allowed team celebrations in any way, shape, or form. This was reflected by the general morale of the team. When the team hits a milestone such as a release, throw a party. Take the team to lunch. Make them feel appreciated.

Facilitating Self-Organization and Cross-Functionality

When you ask a team to be cross-functional, you are asking them to set aside some of their expertise to become more versatile. In the past, team members who were experts were valued. Now, expertise is still valued, but needs to be shared. As I like to put it, team members need to become more wide and less deep. There is nothing wrong with being the expert; however, a cross-functional team needs to consist of members who can jump in and work on anything. Having only one person who is the expert on a particular function or module of code is not desirable. What if this particular person hits the lottery tomorrow and is never heard from again? What if they get sick or go on a long vacation and are not available? The Scrum Master needs to encourage the team to become as familiar with every piece of what they are working as possible. Use techniques like pair *programming*, where two developers work on the same thing simultaneously while sitting right next to each other. Usually one developer types as they collaborate together, giving them a shared understanding of the code.

It's the same thing with testing. When I first started in software engineering, the testers tested everything by hand. System time was expensive, especially on the mainframe. This is no longer the case. With the advent of technology such as the ZPDT, systems are more available than ever. Testers need to be of the opinion that constant testing reduces variability. Tests should be automated if possible and run continuously. Test plans should be written as soon as possible in the development process. In my opinion, teams should be taking steps to move toward test-driven development, where the test automation is written first, then the code is written so that it passes the test.

The team needs to move from coders and testers to team members. Developers can test. Can the testers pitch in anywhere and write code? All the team members need to grow in their skills so that the team becomes truly cross-functional. Scrum teams need to become more cross functional so that they can take on more challenges.

The cost of becoming more cross-functional is that team members lose expertise. Encourage team members to attend *community of practice* meetings, where a group of experts in a particular subject get together to discuss ideas and deepen their knowledge. I coach Scrum Masters to get together once a week to talk about Agile methodology. Developers and testers should do the same thing. Get together and let iron sharpen iron.

Coaching Team Members and Developing Their Competence

A Scrum team needs to contain all of the skills necessary to deliver value each and every Sprint. In order to do this, some capacity needs to be set aside to learn new skills—a new programming language, or becoming familiar with a new operating system, or whatever. A Scrum team needs to be able to better themselves. Too often, the team concerns itself with burning down the Backlog and forgets about preparing for the skills of the future. There is no capacity put aside to get better. The Scrum Master needs to remind the team that professional development is important and ensure that the Product Owner understands that this is a situation where we are sacrificing a bit of velocity now to go faster later.

Promoting Team Members' Career Advancement

The Scrum Master should be an advocate for the Scrum team and its members. I've been told by various people in upper-management positions that they take great pleasure in seeing their people do well. Part of being a servant-leader is putting the team's interest first. To use a hip hop term, the Scrum Master is the team's *hype man*.

A hype man is the backup rapper whose job it is to support the primary rapper. The job of the hype man is to get the crowd excited and keep them that way. Editorialize and socialize everything the Scrum team members are doing. Encourage team members to go after new positions and to enhance their careers. Do whatever you can do to help each team member archive professional success.

Celebrate these successes. After all, when the team wins, we all win…

Stepping In and Stepping Back

One of the things I struggle with is when to stop a conversation and when to let it continue. As a meeting facilitator, I want to make sure that the meeting is valuable to the whole team and does not run over the agreed-upon time box. As a Scrum Master, I want the team to collaborate and have valuable face-to-face conversations.

What I try to do is recognize when this is happening and ask the team what they want to do. If they want to allow the conversation to continue, I'll set a five-minute timer. At the end of five minutes, I'll ask the team if they want to continue the conversation for two more minutes. I'll continue these two-minute time boxes until the conversation is over, or the team wants to continue in a parking lot.

Pitching in

If I haven't been clear enough to this point, allow me to say this as clearly as possible. Scrum Masters do not, under any circumstances, write code. A Scrum Master exists to serve the team, lead the team, and challenge them to become more Agile day after day. Yes, a Scrum Master is a member of the team. They are not engaged in helping the team do their work.

Keep the team focused on inspecting and adapting.

Keeping the Team Motivated, Energized, Empowered, and Cohesive

There is real power in positivity. You may dismiss the power of positive thinking, but I am here to tell you that it works wonders. For example, I used to be happy to talk about folks getting hit by a bus. I'd say things like, "What if Gene gets hit by a bus tomorrow? Who can pick up his work if you don't understand what he's doing?"

On the surface, there is nothing wrong with that statement. If Gene gets hit by a bus, the team isn't going to see him for a while. That's undeniable; however, it's also negative. Now, I prefer to put it this way: "What if Gene hits the lottery, buys his own private island, and we never see him again? Who can pick up his work if you don't understand what he's doing?"

That sounds a lot better, doesn't it?

I like to describe myself as "sickeningly positive." I'm usually a glass half full type of guy anyway. I just turn it up a bit. It rubs off on the team.

Think about what is coming out of your mouth, because it impacts not only those around you, but your attitude as well. I try to not use words that threaten success like might, maybe, can't, won't, if, …… I try to catch myself saying them and replace them with words that will make success possible.

Shielding the Team from Interference

One of the Scrum Master's primary duties is to shield the team from outside interference so that they can focus on achieving Sprint goals. This is easier said than done. I like to coach Scrum Masters to be filters. Everything that goes to the team must first pass through the filter, the Scrum Master. When I say everything, I really mean *everything*. For example, salespeople get pressure from customers to get that one feature added to the product before they will buy. Support reps want to ask questions. If there is a proof–of-concept event at a customer site or some type of demo, people in the field want development involved.

I'm not saying that a Scrum team should ignore requests such as these. Not at all. What I am saying is that everything a Scrum team does must be put in the Backlog and prioritized. The team makes a commitment at the Sprint planning meeting. This is the amount of work they can deliver. It's not fair to add work to what is already in the Sprint Backlog.

Everything that the Scrum team works on needs to be vetted by the Product Owner. Once the Product Owner understands the request, it can be put in the Backlog and prioritized.

What if what is being asked for has a super-high priority? That's fine. However, if the team needs to work on it right now, something needs to be removed from the current Sprint. Nobody gets to pile more work on the team.

Educating the Organization

A mistake that a lot of organizations make is to focus on educating the Scrum teams. At first glance, this is a good idea. Scrum teams should know Scrum. That makes sense. However, to change the culture, the whole organization needs Agile training. Even though folks aren't directly working on Scrum teams, they need to understand Scrum. As I like to say—everybody needs to speak the same language.

A Scrum Master should be doing webinars, blogging, creating educational videos—using any media available to educate everybody about the benefits of Scrum. The entire organization needs to understand the Scrum framework, because the entire organization needs to become more Agile, not just the development teams.

In this chapter, we looked at some of the soft skills required by the Scrum Master role. From breaking up fights to lightening the mood to shielding the team from interference, these soft skills are an important part of keeping the team happy and content. In the next chapter we will explore the technical skills of the Scrum Master.

CHAPTER

9

Technical Skills of the Scrum Master

The Scrum Master isn't considered a technical role. In fact, I'd argue that nontechnical people make the best Scrum Masters because they can focus more on Scrum itself and the people skills we discussed in the previous chapter. There are some technical skills, however, that need to be in the Scrum Master's wheelhouse. We will look at them in this chapter.

Creating a Definition of Done

The most important thing to remember is that you have to start somewhere. Far too often, I coach a team through the creation of the DoD or DoR and it turns into frustration. Paralysis by analysis—teams have a tendency to overthink things like this. One way I like to keep teams from overthinking things is to use the sticky note exercise. Everybody on the team gets a sticky note and pen. I ask each team member to write down something they want to see in the DoD on a sticky note. I then collect them and put them on a flip chart. I then remove duplicates, and discuss what is left. If the team wants to remove anything, there has to be consensus. Once we get through the first set of sticky notes, we go through a second round. After a few rounds, you should have a workable DoD.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_9

The DoD should be a living document. A team should not create the Definition of Done and then never tweak it. For example, something may arise from the Sprint Retrospective that requires the DoD be modified. I strongly suggest that the Scrum Master revisit the DoD with the team every few Sprints or so to ensure that nothing needs to be added or removed.

Selecting Team Tools

As a Scrum Master, I try not to get hung up on team tools. Personally, I'd rather use sticky notes on a wall. It's a simple solution, but it works. I also understand it's not realistic, especially if the team is not all in the same place.

Let the team pick what they want to use. Experiment and figure out what's best for your team and your situation. There will always be factors external to the team (mainly budget) that will influence the final decision. The bottom line is that a tool is just that—a tool. All the bells and whistles in the world won't help if you can't use the tool to accomplish what you need to do.

Building Design and Architecture into the Product Backlog

One of the Agile principles is, "The best architectures, requirements, and designs emerge from self-organizing teams." Both design and architecture are viewed as emergent in Scrum. That means we recognize that this stuff will emerge as stories are worked on. Do just enough design work up front and add architecture in appropriate stories that are added to Sprints as needed. This causes people to freak out because they feel that the engineering aspect of development work is being ignored.

Engineering should continue throughout the development cycle. Agile does its best not to have long-running research and design up front. Get to building stuff sooner and figure out the architecture as you go. In Scrum, we use phrases like "minimally viable" and "maximize the amount of work not done." We simply don't like doing a bunch of work up front that we will probably have to throw away.

Emergent architecture comes from within the team as a need is discovered. For example, when refining a story, it becomes evident that there is some architecture work to do. The architect should work with the Scrum team to define these architectural requirements and get them in the Backlog as soon as possible. The Product Owner will prioritize them with the other work.

The architect should work with the Scrum team as architecture emerges. He or she shouldn't write code per se, but should work with the team to ensure that design is consistent and meets architectural standards.

Refactoring

Refactoring is changing the internal computer code without changing the external behavior. At the risk of being simplistic, I explain refactoring as making the code better while not changing the behavior of the product.

I love to cook. I love to eat as well, so I guess my love of cooking goes hand in hand with that. While I think I'm a pretty good cook, I am terrible at cleaning up after myself. I get all engrossed in cooking and the next thing I know, I'm surrounded by dirty pots and pans and all kind of messes to clean. My mother-in-law is different. She cleans as she goes and doesn't have to stop at the end to spend a lot of time cleaning. That is code refactoring.

The goal is to keep the code clean and easy to read. It's not about fixing bugs or rewriting things. Refactoring is about doing stuff like removing any duplicate code during a coding session. We don't want the code to become big or unwieldy as we go through iterations or adapt to changing requirements. As we add to the code, we continuously refactor it to remove complexity.

Continuous Integration

Let's say that your team has just finished a Sprint demo. The customer is so impressed with what your team has done that they want it now.

Yes, you have built a minimally viable product, but the customer isn't asking for that. They want full-blown product. In other words, a release.

Delivering a minimally viable product every two weeks can be difficult to achieve. It may require the Scrum team to change the way they go about their business.

The ultimate goal of continuous integration to be able to deploy at any time. To keep the product build, tests, and other release infrastructure up to date as much as possible.

Most teams that I work with have a trunk or main repository that holds all of the source files for a particular project. Team members "check out" or make a copy of the module in the trunk and copy it to their local workstation. Once the modifications are complete, the developer "checks the file in" or copies it back into the source repository. Now the file in the source repository should contain all of the developer's changes and match the file on his or her workstation.

Usually there is some type of source control in place that will not allow a source file to be worked on by more than one team member simultaneously.

Merge (or integrate) the team's code changes into an integrated source repository daily. The more time that the team waits between merges, the harder it is to find issues.

Continuous integration takes things a step further. When something is checked into the source, a build is automatically kicked off. Once the build completes, the new product is automatically installed and configured on a continuous integration test machine, and automated test suites are executed against the newly built software. The entire process is "lights out," meaning that it is automated entirely. This way, every change is tested thoroughly. The team can have confidence that everything that is checked in is not only thoroughly tested, but works together. It goes without saying that if the process fails, the whole team stops to fix it before anything else is done.

Automate the Build Process

Most programming languages such as C or C++ require the code to be compiled. You start with binaries or source files produced by the Scrum team and end up with deployable software. In other words, human-readable source files are converted into machine-readable objects that the end user can install. They include things like .exe and .dll files on a Windows computer. A simple build process looks like this:

- Compile the code.
- Link the objects.
- Create the installation package.
- Run reports.

A build can be automated, manual, or a combination of the two. A manual build requires that commands be issued and executed one by one. Most projects include many source files. A developer can compile each of these files individually. It's an understatement to say that manual builds can be tedious.

Instead of manually building everything, Scrum teams can automate the build process so that the entire project can be built with one command. My teams called this a "make." The program goes through a list of source files, called a *makefile*, and calls the compiler for each one. Sophistication can be built into the make so that only the changed files are compiled, or so that other programs can be run after the compile. Automating the build process makes continuous integration a reality. Being able to build with a push of a button or to have an event such as somebody checking code into source management kick off a build enables rapid deployment.

Test-Driven Development

The idea here is that no code can be trusted. At first that seems a bit harsh, but when you look at things, it makes sense. *Test-driven development* (TDD)

is where the team writes small specific tests for specific stories. The test is written before any specific coding is done for the story. When the test is first run, it fails. The code is then written to pass the test and no more.

Acceptance test-driven development (ATDD) is more of a collaborative effort. It involves the development team, Product Owner, and even people from the business. I've even heard of customers getting involved.

The acceptance tests are written before any source code is written. When the code is written, it is written to pass the test. To be clear, I am not talking about acceptance criteria here. The acceptance criteria may be used to write the acceptance test, but the acceptance test shouldn't be solely based on it. The team should take everything into consideration. This shouldn't be too hard to wrap your mind around. Before a line of code is written, the team looks at the following:

- Story personas
- Use cases
- Acceptance criteria
- Whatever else they believe is needed to provide the value asked for

A very detailed acceptance test is then written. This test ensures that the proper value is delivered through whatever testing scenarios are defined. This seems to turn the world on its head, as the focus is now on writing the test. The code is written just to pass the test. The team does not write any production code unless they have a failing test in place.

TDD is all about making sure the code works; for example, a function gives the proper return code. ATDD is more about results; for example, the user clicks on a link and is taken to the correct page.

It's going to feel uncomfortable, weird, and awkward to the team at first. They will not be used to working in this fashion. You really need to show the value of TDD and get the team to buy in.

These techniques almost guarantee that there will be no defects in the production code. That sounds pretty valuable to me.

Updating the Release Plan after Every Sprint

The Release Plan is a document that shows what the team plans to deliver to customers and stakeholders. It sounds simple enough. Just tell the customer all of the wonderful features the Scrum team is going to build, and get to work…

Not so fast. Remember, we need to plan for variability as we build software. As a team, we need to expect change.

"Pick your opener so that you are in the meet."

I heard my powerlifting coach Jake Turner say that hundreds of times. He was talking about planning attempts in a powerlifting meet. I think it parallels nicely with how to set up a release plan for an Agile project.

In the sport of powerlifting, a lifter performs three lifts: the squat, bench press, and deadlift. Each competitor gets three attempts at each lift. The highest successful lift in the squat, bench press, and deadlift are added together. The highest total amount lifted in each weight class wins.

If a lifter does not perform a successful lift after three attempts in any one of the three lifts, they are disqualified and their total does not count. This is referred to as "bombing out" of the meet.

I only bombed out twice in my powerlifting career. In both cases, my first attempt was too heavy. In one case, it was my deadlift. I was so tired after squatting and bench pressing that I couldn't get the bar to budge off the floor. The other time I couldn't hit depth in my squat attempts. The judging was tough, but not unfair. I should have gone lighter on my first squat attempt. I don't think it's a coincidence that both times I bombed out, Jake was not coaching me.

The rule that worked against me was that once a lifter called for a weight, it could not be reduced. In other words, if a lifter called for five hundred pounds on their first attempt and missed the lift, he or she could only repeat five hundred pounds or go up in weight. Jake would continuously say that you should pick your opener to stay in the meet. In other words, a lifter should be super conservative with their opening attempt at each lift. He would encourage me to pick a weight that I could easily do for three repetitions. By picking a light weight for an opener, the lifter is accounting for variability. The equipment at the contest is different than what the lifter is used to. The lifter probably had to travel to the meet. The lifter may have had to cut weight to lift in a specific weight class. The judging could be strict, or even unreasonable. Sometimes lifting in front of a crowd of people causes anxiety.

All of this stuff can affect a lifter's performance. I'll give you an example. Every time a certain judge (who shall go unnamed) was working a meet, I knew I wasn't going to get a fair pause when performing a bench press.

Powerlifting rules state that in the bench press, the bar must rest on the lifters chest until it stops. This is done to keep the lifter from bouncing the bar off his or her chest and using momentum to complete the lift. The head judge would watch the descent of the bar, and give a verbal "Press!" command to signal the lifter to complete the lift. That's the rule and I trained my bench

press accordingly. However, as sure as the sun was going to rise in the east, I'd wait three or four seconds for the press command if this particular person was in the head judge's chair.

Jake's rule was that you open light, your second attempt is what you felt you could reasonably lift that day, and your third attempt was for a personal record attempt or a weight you needed to win. If I had opened with anything more than an easy triple (a weight I could do for three), the long pause would have probably caused me to miss that attempt. To be honest, every judge was different. Sometimes the command came so quickly it surprised me. Sometimes I felt like asking the judge if he fell asleep. That is a good example of how variability can negatively affect even the best-laid plans. A lifter would usually take from ten to twelve weeks to prepare for a contest. That's quite a bit of wasted effort if you bomb out.

When talking about a release plan, refer to the iron triangle as shown in Figure 9-1.

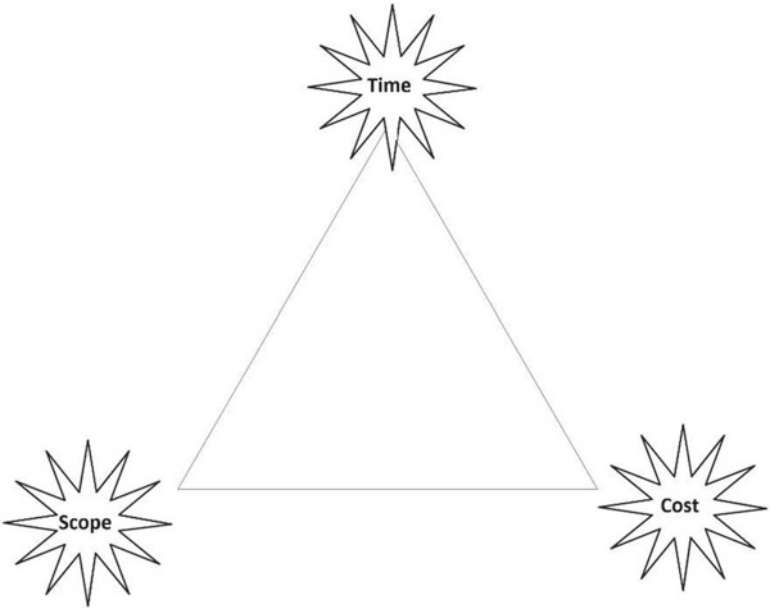

Figure 9-1. The iron triangle consists of time, cost, and scope

The iron triangle is a diagram that shows three factors that affect any project—time, cost, and scope. When talking about Agile projects, these are the only three things you have the power to change. For example, suppose you have a project in flight and all of a sudden you find out that it has to be completed by a specific date. The only two options you can adjust are the cost and the scope.

Think of a boat, a big boat sitting on the water. One of those Viking long boats with oars, or a Greek trireme. So, there are a bunch of guys seated on the boat that work the oars, and they determine how fast or slow the boat will go by how fast or slow they row. The boat also has cargo that has to reach the destination. The guys sitting at the oars are the Scrum team working on a project. They represent the cost factor of the iron triangle. People cost money.

The guys working the oars are what propel the boat. You can't ask them to simply row faster to decrease the amount of time it will take to reach your destination (or finish the project). The same thing goes for a Scrum team. You can't ask them to code faster or work harder. Well, you can… they will work weekends and insane hours for a while, but they are going to burn out eventually.

The burnout will not be pretty. This is what we saw in Waterfall. Teams would do whatever it took to get a release done by the promised GA date, and folks were miserable… This is not what we want for our teams. If you want the boat to get to its destination faster, you really only have two options. You could add more oars and more guys to row. This bigger "engine" will propel the boat faster. The second option is to throw some of the cargo off the boat and make it lighter (reduce scope). With a lighter load, the crew can propel the boat faster without having to work harder.

Any project has three factors—scope, cost, and time. This is reflected in the eighth Agile principle—"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."

When you try to define the exact level of scope, exact cost, and exact time on a project, you guarantee failure because there is no flexibility should something change. You are not accounting for variability or expecting things to change. One of the reasons we are doing Scrum is that we recognize that it's impossible to plan everything out at the beginning of a project.

Notice that quality is not changeable. Neither is the Definition of Done.

Just like picking an opening attempt in powerlifting, don't make your Agile release plan so inflexible that any type of change or variability will disrupt the release. When building a release plan, start with epics. Epics are groups of stories that have a common theme and deliver a feature or features that a customer would want. For example, if I was building a commercial website,

an epic might be accepting payment. Another might be the shopping cart, or browsing products. The Product Owner and team will give the epic a very rough size. I call these SWAGs (Scientific Wild Ass Guess). Normally, a SWAG will be pretty big, and it should reflect the relative size of the epic. Just like story points, but on a much bigger scale.

Epics fall into one of two groups: definite and nondefinite. A definite epic is something we plan to deliver in the release. We socialize the definite epics with customers and stakeholders. These are the things that we want them to get excited about so that they partner with us and get involved in the development process. When setting up a release plan, definite epics should make up 50 percent of the total work defined in the release.

The team, Product Owner, and business should agree to how much work or capacity the team can handle for the upcoming release. Factors such as average team velocity, the length of the release, and the general business climate should be taken into consideration. Usually, cost is also a big factor. You want to balance the cost and return on investment (ROI) when trying to figure out how to build a release plan.

Nondefinite epics should make up around 20 percent of the release capacity. These are epics and stories that are likely to be done, but are not promised. This is one of the ways the team plans for change. The customers and stakeholders are not told about this work. Nondefinite work may get done, or it may not. By nature, it is expendable. It's nice to have, but not critical. Definite work should be completed before any nondefinite work is considered.

So, the scope of the release should be right around 70 percent of the release capacity. What about the other 30 percent?

The other 30 percent is for working on technical debt and team improvement. Remember that transparency is one of the three pillars of Scrum. We want the team to be able to better themselves by learning new programming languages or learning about upcoming operating systems or development techniques. We also don't want to ignore technical debt or the refactoring of existing code. Instead of hiding it or expecting the team to figure out how to fit the work into a maxed-out schedule, we put it right in the release plan. My friend and fellow Agile coach Jay Cohen likes to see 10 percent put aside for team improvement and 20 percent for technical debt. Every team is different, but I like to see at least 30 percent of release capacity focused on something other than the release itself so that the team has the ability to realistically address deficiencies. Figure 9-2 shows the breakdown.

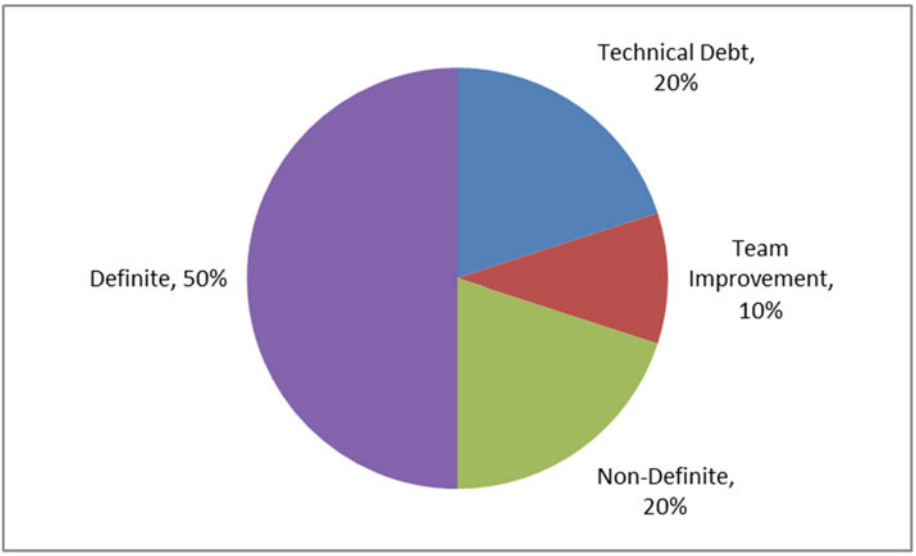

Figure 9-2. Epics, debt, and team improvement

At this point, start to build a timeline. Decompose the highest-priority epics into stories and fill out the first two or three Sprints, as shown in Figure 9-3. Use the team's previous average velocity to figure out how much work to schedule for Sprints. If you don't know the team's average velocity, make your best guess. The Scrum Master will recalculate the average velocity after the first three Sprints and every Sprint thereafter, so that number will probably change.

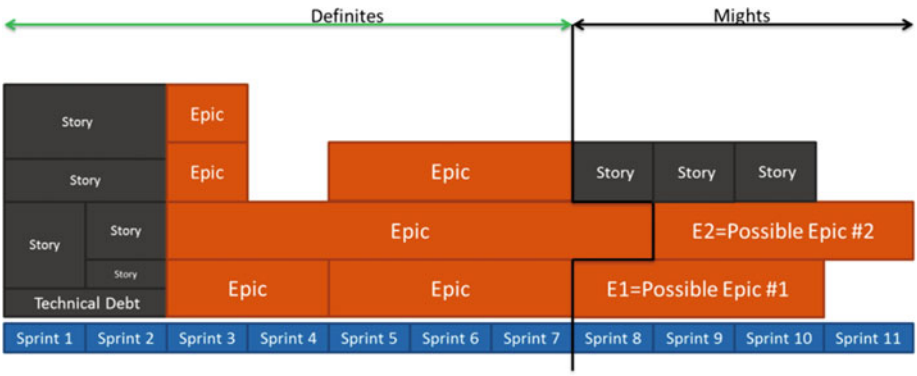

Figure 9-3. The release plan timeline

Once the first three or so Sprints are populated, try to roughly fit the rest of the definite epics into the rest of the Sprints. By having only 50 percent of the release filled with definite work, the team has "wiggle room." We only want to have two or three Sprints populated with stories at any point during release work. Going any further into the future is a waste of time and work. We expect things to change due to customer feedback and changing requirements. If customer feedback changes the scope of one of our definite epics, we have the flexibility to do that work, satisfy the customer, and still deliver all of the definite epics, and the same thing if changing requirements force us to add something to the release.

Prioritizing Items

I keep saying that the Product Owner should be routinely prioritizing the Backlog. How does he or she go about doing this?

There are a bunch of ways to do this. What I like to do is coach the Product Owner to think about three things: business value, complexity, and the cost of waiting.

I coach Product Owners to prioritize stories that deliver the biggest ROI. Balance what you can get done quickly with what customers want quickly. Weigh the cost of delay. You obviously can't do everything right now. What can wait and not adversely impact the product? What needs to be done right now to ensure that customers are happy and maintain the business?

Don't forget about complexity. A story may be high on the priority list, but very complex. Does it make sense to work on a less complex story that is also high in priority but will take less time to complete? Remember, it's about maximizing ROI.

Running a Sprint Planning Meeting

The Sprint Planning meeting is where stories are brought into a Sprint, decomposed into tasks, and assigned to team members. The team agrees to complete a certain amount of stories in the meeting and then plans how to deliver them.

I like to plan out a team's capacity in "buckets." Take each team member and determine what their availability is for the Sprint. Remember, we use ideal days. I like five hours per day for each team member. Don't forget about things like holidays and vacation. Now you have a number. For a two-week Sprint with no time off, each team member should have about fifty hours in their individual bucket, as seen in Figure 9-4.

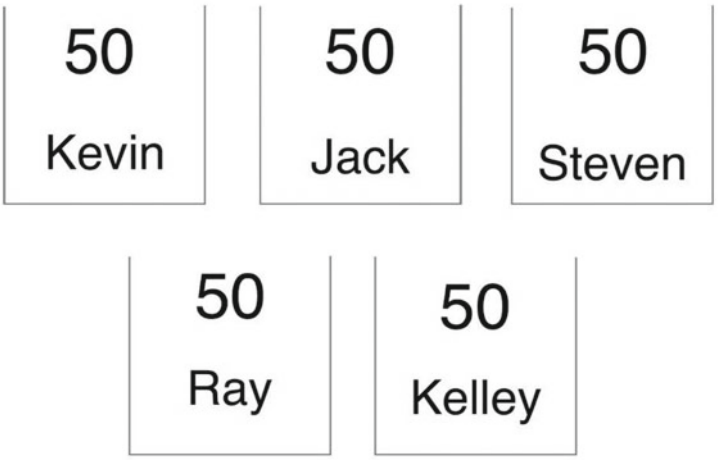

Figure 9-4. Individual buckets

Next, I reduce each team member's bucket by ten hours for meetings and Agile ceremonies as seen in Figure 9-5. You have Sprint Planning (the meeting you are currently running), Backlog Refinement, Sprint review, Retrospectives, any other team meetings and things like town hall meetings and such. All of those meetings really don't add up to ten hours. I like a bit of a buffer as well.

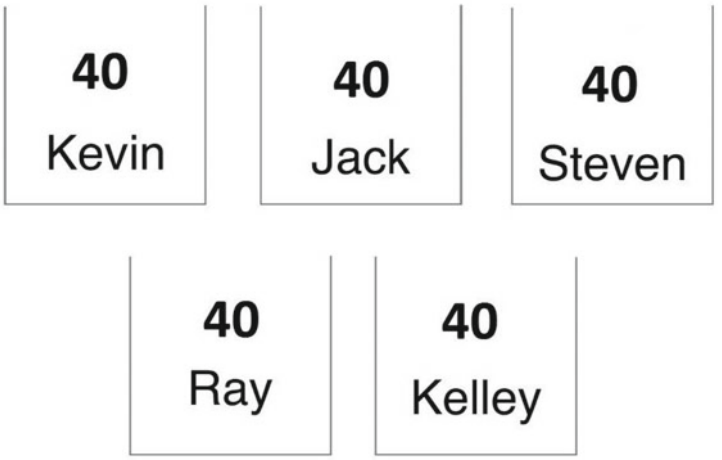

Figure 9-5. Reduced buckets

Why?

I'm planning for variability. I'm leaving room to change plans if need be. If you overfill the Sprint, any type of change will make the team unable to complete all of the work in the Sprint. If support needs help with a high-priority issue or a team member gets sick, the team has room to adjust and still complete the Sprint work.

Obviously, if a team member can't work every day of the Sprint because of vacation or a trade show, or something like that, their bucket gets reduced as shown in Figure 9-6.

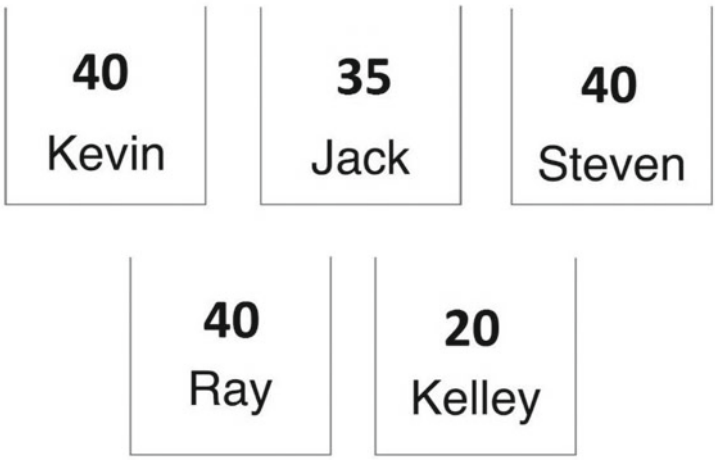

Figure 9-6. Kelly and Jack are taking time off this Sprint

Now you have the team capacity figured out. For a five-person Scrum team, the total team capacity is 200 hours for the two-week Sprint as seen in Figure 9-7.

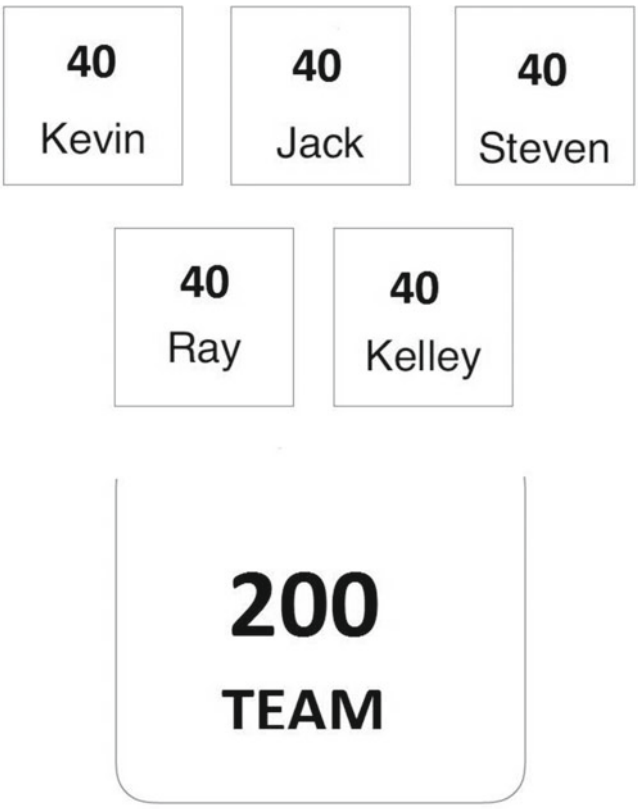

Figure 9-7. The team bucket

At the beginning of the meeting, the Product Owner should lay out the Sprint goals. Then the team picks the highest-priority story and decomposes it into estimated tasks.

Decomposing Stories and Tasks

In the Sprint Planning meeting, the team takes a story and decomposes it into tasks. Be conservative with estimates. There is nothing more important than keeping task estimates reasonable. Remember that Sprint commitments must be kept. If a story is too big, split it. Don't be afraid to return a story to the Backlog for further refinement if necessary.

As the team works on decomposing stories into tasks, coach them to refine those tasks as small as possible. As I said before, don't just create a task for development, a task for testing, and a task for documentation. Try to decompose tasks into development steps. This will help the testers and

documentation folks get involved in the process earlier. Of course, if the team is doing test-driven development, that isn't an issue. Most teams aren't, so they suffer from what I refer to as "hockey stick Scrum," This is where the testers are expected to do all of the testing the last few days of the Sprint. The hockey stick leads to stress on the testers and stories not being completed in the Sprint.

Don't out-kick your coverage. Have you ever seen this? You are watching a football game and the punter launches a really long kick. You would think that this is a good thing; after all, you want to back the other team's offense up as much as possible. However, what usually happens is that the punt gets returned for a touchdown. Why? Because the coverage team could not get down field fast enough. The returner was able to back up, catch the ball, and have plenty of time to identify blocks and exploit seams in the kick coverage.

The same thing can happen on a Scrum team. Developers want to write code. I get that, but there is more to being done with a story than being code-complete. Stuff needs to be tested and documented. Hockey stick Scrum causes the testers to get swamped with work at the end of Sprints so stories don't get done. Essentially, we out-kick our coverage.

The big issue here is how both developers and testers worked under Waterfall, which dictated that you tested at the end of the project, and you wrote your test cases from the design specification document. Trying to work this way within a Sprint does not work well at all.

The way around this is to have testers involved throughout the development process. The team needs to do whatever it takes to get testing done as early as possible. The test cases need to evolve while the story is being worked, and it may require that testers be given stuff to test as the story evolves—not when it's code complete. This requires tight collaboration between the tester and the developer. They have to (gasp!) actually talk.

The cool part of this is that development becomes keenly aware of the testing effort, so they can give some great feedback on writing effective tests. Also, the "final" testing on stories becomes much easier because testers were very familiar with the story. Lastly, the team gets a lot better at sizing stories, because of a deeper understanding of the effort required to get to done.

Yes, I know that nirvana for a Scrum team is for there to be no established roles… you know, no more developer, QA, or whatever. Everybody codes, everybody tests, whatever it takes.

As a team, do not put yourself in the position where you can't deliver what you have committed to. It's always better to under-promise and over-deliver.

Refining Estimates

A Scrum team should never change the size of a story. OK, maybe never is too strong a word. A team should rarely change a story's size. Sizing a story is a team activity and is relative in nature. If this story is a three, how big is this other story in relation to the first one? As the story is defined, the team better understands what the Product Owner is asking to be delivered and the work required. In this case, the size may change because the team better understands the work.

That, however, is not something that happens a lot. A more frequent scenario is where the team sizes a story incorrectly. Normally, the size is too small. The team starts working on the story and finds out that there is a lot more work to do than originally thought. For example, a story that the team originally thought was a three ended up being a thirteen.

In this case, the team is tempted to go back and resize all of the other stories in the Backlog. At the surface, this looks like the right thing to do.

It's not.

Story points are used to calculate team velocity. The funny thing about velocity is that it works itself out. I like to say that it all comes out in the wash. Even if you grossly over- or under-size a story, the average velocity will still be accurate. If a story that you thought was a three turned out to be a thirteen, the team's average velocity will tick down slightly but correct itself over time. Even if all the stories the team thought were threes turn out to be thirteens, velocity will still accurately predict when the team will complete the Backlog. An exception might be if the team finds that a story needs to be split. New estimates would need to be given for the resulting stories. Ideally, this will happen before Sprint Planning, but it might happen in Sprint Planning. Either way, once you start working on the story, don't look back!

Sizing and estimating are two different things. When the team decomposes a story into tasks, each task needs to be estimated. The team comes to an agreement on how much time it will take to complete each task. This is called *estimation*. Different teams estimate tasks differently. I've seen the team come to consensus on how long a task will take, and I've seen individual team members size tasks by themselves. Both approaches are acceptable in my eyes. The key is that all of the tasks are estimated. If a team member is working on a task that was originally estimated at 6 hours but turns out to be grossly underestimated, increase the hours on the task. If it was originally was estimated at 6, but now needs 20 hours to complete, change it.

The Sprint burn-down chart is used to gauge all of the work remaining in the Sprint and how quickly the team is completing it. The team needs to not only "burn down" task hours as they complete them, but also give a realistic

picture of how much work is left to complete the task. Unlike story points, burn-down estimates should be as accurate as possible. If the estimation is incorrect, change it so that the Sprint burn-down is accurate. If there is a problem, the team, the Scrum Master, and the Product Owner want to be able to see it well in advance.

The reason we want to see things clearly is to take action before a problem arises. If the burn-down shows that the team is not going to be able to finish all of the tasks before the end of the Sprint, the Product Owner and Scrum Master should take action. Maybe the Product Owner can reduce the scope by moving stories out of the Sprint. The Scrum Master should find out what happened. Was it really just the estimate being incorrect, or is there something else to dig into? This is not like the old Waterfall status meetings where the manager looked for reasons to chastise people for being behind. The Scrum Master, Product Owner, and manager should be looking for ways to help the team to meet their goals. The team needs to work at a maintainable pace.

Handling a Defect Found in a Sprint

A team wants to honor its Sprint commitments. What happens when the testers find a defect? Worse yet, what happens when a defect is found late in the iteration?

You fix it.

I coach teams that the highest priority Sprint work is fixing defects. Too often, I see defects pushed to the back burner. This is a process I call building a defect mountain. Eventually, these defects will have to be addressed, and it will be painful.

I'd rather see a team not deliver all the promised stories in a Sprint so that the defects can be fixed. It's that important. The rule is that you fix defects in the iteration they are found in. If stories need to be removed from the Sprint and put back in the Backlog to account for the time needed to fix defects, so be it. Sometimes it's helpful to look at a situation from a different perspective to fully understand the impact.

My wife bought me a fitness tracker for my birthday. You don't see many 300-pound, 80-year-old guys walking around, so my focus has switched from moving big weights to getting moving. The fitness tracker is a gadget that tells me stuff like how many steps I take in a day, calories burned, and how many miles I've traveled. I found myself checking the thing constantly and striving to reach my daily step goal.

Then it quit working.

To say I was not happy was an understatement. The thing was only four months old. I contacted the company and they replaced the device. However, I never regained my original enthusiasm for using it. In fact, I recently replaced it with another brand.

Zero defects means just that. Zero defects. Our customers expect what we build to give them the value they expect and to, well—work. Quality is more than a slogan. It is a commitment to our customers to do our very best to deliver working software.

In this chapter, we took a look at some of the technical skills required of the Scrum Master, such as automating builds and putting together release plans. In the next chapter, we dive into some of the contingency skills a Scrum Master may need.

CHAPTER

10

Contingency Skills of the Scrum Master

I grew up in the seventies, back when we didn't use seat belts—we barely used car seats. Riding around in the back of a pickup truck was not thought to be a dangerous activity. One of the things I miss the most about the seventies is the television programs. Maybe it's because we only had three channels (four if you were good with aluminum foil). One of the things I remember from those programs is quicksand. I personally have never experienced quicksand, but it was everywhere on television. You'd be minding your own business, and the next thing you know you are in quicksand. As I remember things, the more you struggled, the faster you sank. You had to call for help. Somebody needed to pull you out.

Nostalgia aside, I think project impediments are like quicksand. When I say the word "blocker," I'm sure you think of a giant rock in the middle of the road. Sometimes impediments are more like quicksand. You think, "I can handle this," but the more you struggle, the more you sink. Get help before it's too late.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_10

Identifying and Removing Impediments

When an impediment is raised, the first thing I ask is: "Is this really an impediment or something the team can handle?" If a team member is sick and will miss the rest of the Sprint, can the team pick up his or her work? Can the team reorganize and still reach the Sprint goals? Now, if that team member hit on a huge lottery ticket and bought their own private island somewhere, that is an impediment. We need to replace that person on the team or plan for a velocity reduction. Something becomes an impediment when it exceeds the capability of the team.

When dealing with an impediment, it's useful to understand its scope. In other words, who has the authority to handle it? As a Scrum Master, you don't have much authority. Understand who can help you with the problem and get them involved as soon as possible.

Once an impediment is identified, I like to put it on my impediment board. An impediment board is an information radiator that is put in a place where everybody can see it. This way, there is no question about the state of any impediment the Scrum Master is working on. It is also a place for people to look and see if they can help out with anything the Scrum Master is dealing with.

I also coach my teams not to wait for the daily stand-up meeting if the impediment is urgent. Don't let protocol allow you to sink in that quicksand. Get the Scrum Master involved as soon as possible.

Changing Team Composition and Personnel

In a word, no….

The Scrum team is where the magic happens. I've said before that there is nothing a Scrum team can't do. To me, the team is the cornerstone of Scrum. I don't like messing with the team. Adding or removing members is disruptive and should be avoided if possible.

Instead of disrupting the team, move work to the team. If the work is important enough to break a team up, consider having the whole team do it. Moving work to the team enables the team to stay together and the business to meet its needs.

Preparing and Running a Sprint Review

The Sprint review is the Product Owner's show. Yes, the team gets to show off all the cool stuff that they built, but the PO builds his or her relationships with the customers. They get to highlight that the customer value is being realized.

I like to facilitate the Sprint planning meeting for the PO. It allows the PO to focus on the customers and stakeholders and not have to worry about the overhead of running the meeting. I like the PO to know that I have their back. This is one way I do that for them.

The focal point of the meeting is the demonstration of what has been built, so make sure that you have that lined up first. Ensure that you know which team member is going to demo what piece of functionality. No, I don't do the demos. I don't have the technical expertise and probably would not give a very good demonstration of the functionality. Plus the team member gets to "show off" a bit.

Make sure that the planned demo is in good working order, and does not show anything that is confidential or company-specific. Make sure you aren't showing off more than you want to.

That being said, don't be afraid to show off the "sausage-making" process. These customers and stakeholders are part of the development process. I don't expect the demos to be polished like something the company would be showing at a trade show.

Next, either the PO or I build a small slide deck. What!?! Mister "I hate PowerPoint and wish we would cancel our corporate license" wants to build a slide deck?

What I hate about the way folks build slide decks for Sprint reviews is that they go overboard. All I want to show is where we are in the release plan—what stories are done and what stuff we plan to work on in the next iteration. I might throw in another slide in that lists who is on the Scrum team and what role they fill. That's it.

As for the meeting itself, I open the call and greet the customers on the phone as well as anybody in the room.

Yes, in the room. You want to encourage customers, stakeholders, team members, management, and anybody else you can think of to attend the meeting in person. Realistically, most customers won't be local so you will need to use technology to show them what's been built.

I usually do a customer roll call so we understand who showed up, and introduce the team. The PO then explains where we are in the release plan and introduces the demo. After each demo, ask for feedback.

At the end of the meeting, I make sure to thank everybody for coming to the meeting and let them know that if they have any additional questions or feedback, they can contact the PO or Scrum Master.

Facilitating Virtual Meetings

It goes without saying that we value face-to-face communication over anything else. That isn't always possible, so how do you ensure that a virtual meeting is valuable to everybody involved?

This is where the Scrum Master's facilitation skills need to be on point. Encourage over-communication at all times. I often find myself asking, "What input do those of you on the phone have??" Make sure that you take every opportunity to include everybody (both local and remote) in the conversation. If it makes sense, use webcams so that you can see the remote folks and they can see you. Not being in the same room can be a limitation; however, it can be overcome with proper facilitation.

Coordinating the Work of One Product Backlog with Multiple Teams

To put it bluntly, successfully coordinating multiple teams on a single product Backlog comes down to commitment and managing Work In Progress (WIP). If the Backlog is properly prioritized, the teams should have no problem understanding what work needs done. The teams should have their Sprint plan together, and come to agreements as to which stories they will take into their Sprint Backlogs. If there are any inter-story dependencies that span teams, they need to be identified. In this case, the teams need to make commitments to each other to ensure the dependencies are met. Once all the teams commit to the work in their Sprint Backlogs, they need to pay very close attention to their WIP and manage it. It's critical that each team get their high-priority stories done. A Scrum of Scrums meeting, as detailed in Chapter 4, should be held to ensure transparency between teams. The Scrum Masters from all of the teams working from this Backlog should assist with the facilitation of the Scrum of Scrums meeting. At the very least, they should get one started.

Getting Help

A Scrum Master doesn't have to know how to fix any situation. They should know who could. I like to say that a Scrum Master has no authority, because in reality they don't. Know who has the proper authority to take care of the current situation and then don't be shy.

Seriously.

Just as a Scrum team shouldn't allow impediments to fester, a Scrum Master needs to take care of impediments as quickly as possible. Don't let fear or pride get in the way. Ask for help when you need it. Remove the impediments.

Countering Scope Creep

Scope creep is when the project grows beyond what was originally planned. It usually will show up in either the Sprint burn-down or the release Backlog. It starts out innocently enough. A customer asks the PO if "as a favor" they could add something to the Backlog. A well-intentioned developer adds some tasks to a story because she feels it's the right thing to do. Sometimes scope creep comes from something external to the team, like support for new operating system features. Sometimes it's because the project vision is poorly defined or stories were not sufficiently refined.

No matter where it comes from, scope creep can be a killer. Be wary of flat burn-down charts or radical changes in the release burn-up. Both point to stuff being added to the release. As a Scrum Master, talk to the PO and try to see where the extra work came from. When it's identified, take appropriate steps to handle the extra work. If it's due to a communication problem, take the proper steps to ensure everybody understands the scope of the project. If it came from a customer, ensure that priorities are properly adjusted and that this won't inhibit the team from delivering on the definite stories in the release plan.

Reducing Scope

There are plenty of good reasons to reduce the scope of a project. Addressing scope creep is certainly one. Remember the iron triangle. If time or cost on a project changes, scope may need to be adjusted. Maybe a team's velocity is proving to be troublesome. When reducing scope, try to make sure that the team can still deliver the definite stories in the release plan. One of the reasons we don't socialize the nondefinite stories with the customers is to give us the "wiggle room" necessary when a situation like this arises.

If you are required to cut definite stories, this needs to be discussed with the customer as soon as possible. Be transparent and let the customers know exactly how the release will differ from what was originally thought.

Canceling a Sprint

It is not something that happens regularly, but there are times when it's necessary to cancel a Sprint. The Product Owner has the authority to cancel an iteration. Maybe there are unforeseen external circumstances that cause a Sprint goal to no longer make sense. If the work being done in the Sprint no longer creates value, abort the Sprint.

When this happens, the Scrum Master should conduct a Retrospective about the situation. What caused things to go south, and how can the team inspect and

adapt? The PO will have some reprioritization work to do so that the Backlog is correct. The team should then plan another Sprint and get going again.

It's important to keep alignment with the schedule reflected in the release plan, so the team may have to plan for either a shorter-than-usual Sprint or a longer-than-usual one.

Documenting Decisions

The older I get, the less I remember. At least I'm blaming it on old age. As I get closer to my fiftieth birthday, I find myself doing things like walking into a room and forgetting why I went in there. My car keys never seem to be where I left them. When the team makes a decision, it's a good idea to write them down somewhere. This way, they can be referred to when nobody can remember why the decision was made in the first place. I prefer to document team decisions somewhere electronically, for example on a wiki page. Give the whole team access and document everything there. You will be glad you did.

Reporting Team Performance

The program manager will want to track team performance so that he or she can ensure that the project is healthy. They also want to be able to share that information with the sponsor. This information is called *metrics*. Metrics give decision makers information they can rely upon. They reflect the health of the project and development effort. Metrics provide measurements of how things are going, and provide a baseline for measuring the impact of any improvements the team may have implemented. There are several metrics that can be used to help with decision-making.

Sprint Burn-down

The Sprint burn-down shows how quickly the team is completing their Sprint work. It reliably predicts whether all the work committed to in the current iteration can be completed by the iteration end. The Sprint burn-down shows how efficiently the team is working, and if there are any problems that need to be addressed.

Release Burn-up

Similar to the Sprint burn-down, the release burn-up predicts when the team will complete all of the stories in the release Backlog. The release burn-up shows the overall health of the release. It can be used to determine whether scope needs to be reduced to release on time, or functionality could possibly be added.

Defect Trends

How many defects are currently open? How many have been closed so far in the project? Defects need to be taken seriously. Defect trends can be analyzed to ensure that the zero-defect policy is being adhered to.

As I said, there are many ways to reflect team performance. Find what is valuable to your team and the business and be transparent. Nobody is going to hand you a million dollars and say "See you in six months." The business is going to want to understand what the return on its investment is, so they obviously want to know about the health of the project.

CHAPTER

11

Putting It All Together

I'll be the first to admit that when I write, I just document what's running through my mind at the time. At the end of this book, I'd like to wrap things up and share some of my thoughts on what it takes to be successful at Scrum and the Scrum Master role.

Standing in front of a Scrum team can be intimidating. There will be folks that are excited, skeptical, apathetic, and fearful. You are the one the team is looking toward to lead them into this Agile "stuff."

This reminds me of when I was asked to coach my daughter's U6 (under 6) soccer team. My daughter wanted to play, so we signed her up and bought her a tiny set of cleats and shin guards. Before practice was supposed to start, the phone rang. The person on the other side of the line explained that they really needed coaches. When I said that I knew nothing about soccer, the response was, "That's OK, They are kids. Just have them chase the ball."

Being the sucker for a sob story that I am, I agreed to coach. The first practice, I had a bag full of soccer balls, cones, and a whistle. I stood in the middle of a circle of four- and five-year-old children looking at me with those big eyes that children have. I was petrified with fear. I had no idea what to do.

The same thing happened to me standing in front of a Scrum team. Scrum is extraordinarily simple in concept. The practices are straightforward, but actually executing them is very difficult. I expressed interest in being a Scrum Master, and next thing you know I'm assigned to a team.

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5_11

With the soccer team, it was apparent that I had no idea what I was doing, so I educated myself. I talked to other coaches, read books and online materials, and enlisted the help of one of my friends who had actually, you know, played soccer. The kids went from running around like maniacs to doing drills and knowing where to line up. They started passing the ball to each other instead of getting the ball and making a mad dash to the goal.

With my first Scrum team, I was too focused on trying to make them happy. This was a mistake because I allowed compromises that turned out to be detrimental to the team. For example, the team did not see the value of a daily stand-up meeting, so I compromised and held a stand-up three days a week. Why did the team do that? Because they didn't know any better. They were trying to do what they believed. Just like an elephant.

Disclaimer Although I hang around with some pretty big guys, I have never trained an elephant. I have no idea if this next story is true or not. For all I know, it is simply self-help mumbo-jumbo.

How to you keep an 8-ton pachyderm from doing whatever it wants? You clamp its leg to a light chain tethered to a spike driven into the ground.

The biggest land animal on the Earth can be held by a light chain attached to a spike that it could easily pull out of the ground without even thinking about it.

The elephant won't even attempt to pull the chain because it believes it can't pull that spike out of the ground. As the story goes, shortly after an elephant is born, an elephant trainer will take the baby and chain it to a spike driven into the ground. That baby elephant will pull, tug, and try to break free, but eventually it will give up. It will never try to break that chain again—even after it grows to be the biggest, strongest creature that walks.

I guess that's why they say an elephant never forgets.

Many software professionals I know are the same way. They are shackled to the old way of doing things. When you present ideas like test-driven development, or not spending months developing super-technical design documents, or even having a daily stand-up meeting, they don't believe it will work. They don't see the value. They are shackled to that stake.

I've spent the majority of my time as Scrum Master and coach working with mainframe teams. You know, Big Iron. Teams that code in machine language and support products that large companies consider mission-critical. When I first brought up the idea of a four-week Sprint, the response was that there was no way they could write stories that could be done in a Sprint. This is the mainframe. Big legacy code base with all kinds of complex stuff going on.

I'm happy to say that these teams have broken out of the shackles. Today, they are completing stories in their Sprints and seeing the value of Scrum. Most teams are even running two-week Sprints, and their customers are engaged in the development process.

Do me a favor. The next time you say to yourself "that will never work" because of red tape or fear of failure or whatever, think of the elephant who believes he can't break free.

The team I was working with was not trying to be unreasonable. They just didn't trust the Scrum framework. They were conditioned to think that the way they worked under Waterfall was the only way to be successful. There was a healthy fear of change going on as well. As I said before, they used to ask me, "When do I get to write code?"

It was partially sarcastic but did point to a lack of trust in the Scrum framework. In Waterfall, developers were judged on how much code was written. Testers were judged on how many bugs they found. The team found comfort in planning. They weren't trying to thwart my efforts. They honestly thought they were trying to save their performance reviews. The team honestly didn't know what they didn't know. They were looking at Scrum through the lens of "we've always done it this way."

Just like the soccer team, I really had no idea what I was doing when it came to being a Scrum Master, so I immersed myself in all things Scrum. I was a hot mess, so I entered a coaching relationship with the local Agile coach. I networked with other Scrum Masters. I participated in workshops and got training. I sharpened my axe.

Here's another story.

A lumberjack joins up with a new crew. Being the new person, he really wants to impress the team, so he asks the foreman the largest number of trees anybody in this crew had ever chopped down in a day.

"Six," said the foreman.

So the lumberjack went out, put forth a herculean effort, and chopped down a bunch of trees. At the end of the day, he asked the foreman how many trees he'd felled.

"Five," said the foreman.

Encouraged by the fact that he was so close, the lumberjack went out the next day and pushed even harder. Once again, he asked the foreman how many trees he had caused to hit the ground.

"Four," said the foreman.

This upset the lumberjack, so the next day he gave everything that he possibly could muster. At the end of the day, he barely had enough energy to ask the foreman how many trees he had chopped down that day.

"Three," said the foreman.

The lumberjack fell to his knees and put his head in his hands. The foreman put his hand on the lumberjack's shoulder.

He said, "I've never in all my years seen anybody work as hard as you, but let me ask you a question."

"What?" asked the lumberjack.

"These past three days… did you take any time to sharpen your axe?" asked the foreman.

I took the time to learn more about Agile. Scrum team members need to do the same. Set aside capacity during Sprints for team members to better themselves. Sharpen your axe!

Once I got my mind around Scrum and was getting coached, I started to apply what I learned to the teams I was serving. There was so much that had to be done, and I was motivated to be the change agent. My coach, however, asked me to curb my enthusiasm a bit. As he put it, don't go out and try to boil the ocean.

When I went to Air Force basic training, the drill instructors did everything in their power to strip away everything about civilian life. They made us shave all of our facial hair, shaved our heads, and put us all in uniforms without nametags so we looked alike. We slept in the same open bay. We drilled, ate, and went to class together. We no longer had our individuality. All we had was each other.

That doesn't happen with a Scrum team, although I have threatened to make team members do pushups a time or two. When an Agile transformation begins, it seems like everything needs to change. As I said, I was tempted to try go out and change everything at once.

That's not the way to go about a transformation.

So how do you go about implementing Scrum?

The two things that make Scrum work are involving customers in the development process and working to the Definition of Done. Start out by focusing on involving customers and getting the DoD squared away. The most important thing a Scrum team can do is to define a DoD. The acceptance criteria, DoR, and DoD make your team's policies visible. Anytime a story is "handed off," things like DoR and DoD take the guesswork out of the process and make things run smoothly. The team works to satisfy the DoD. This way, there is no question that the story is done. After years of playing games to get releases

out on time with questionable quality, we now ask the team to work stories until they are done. Not code complete. Not "kind of tested." Done.

Start with the basics. Build a Backlog and focus on the Scrum ceremonies. Get the team to agree on a Definition of Ready and Definition of Done. Don't get stuck arguing over the DoR and DoD. Time-box the meeting and just get something written down and posted on the wall—even if you think it sucks. Get the team used to the cadence of Scrum and inspect and adapt. It will get better. Oh yeah, and start having daily stand-up meetings.

Not having executive buy-in and not working as a team will kill Scrum faster than anything. Ensure that those people who have the authority to actually drive change are "all in" with the Agile transformation. Trying to inspect and adapt as a Scrum team in a non-Agile, rigid, command-and-control management structure simply will not work. Scrum requires high levels of trust at both the team and organizational levels. Command-and-control is what gets put in place when you don't trust the people doing the work. I'm sure that there are people out there going through the motions. I just haven't met any of them, and they certainly weren't on my Scrum teams, who had the one thing that I want on any of my teams—pride.

When I was a teenager, Sony Walkmans were the hot new thing, and your music was on cassettes (or vinyl). Compact disks weren't popular yet. My best friend's dad had a Sony Betamax and a bunch of movies he taped off of Ted Turner's new station called "Superstation TBS."

During the summer, I would work construction jobs up until football camp started. The summer of 1983, I was 16 years old, keeping a bricklayer by the name of Tom Aquino in bricks and mortar. It was hard work—really hard work. There is no doubt that I'm doing software work today because I know how hard it is to work construction. It was early August; in fact, if I remember correctly, it was one of the last jobs I worked that year. Football camp was coming quickly, and I would soon trade my work boots for a pair of cleats. We were putting a brick front porch on his house that morning. I was up to my usual antics—mixing mortar and sorting the bricks—making sure that I was only going to give Tom the good ones. Tom was setting up for the job—running strings to make sure everything was plumb and square. "Plumb and square"—it was like a mantra to that guy. I think I heard it a million times that summer alone.

After running his strings, Tom just stood there for at least 10 minutes just looking at the front of his house. The footer was dug and waiting for us, I had the brick and mortar ready, but Tom was standing there.

Finally, I asked, "Are you ready?"

"Let's get rolling," Tom replied.

By lunchtime, we had almost half of the porch done. The left corner was done, with a brick column that went around 6 feet up. So I'm sitting there eating and I'm watching Tom. He's standing in the same spot that he was earlier… looking.

Then the unthinkable happened… He pushed the column over.

To be honest, I thought he'd lost it. "What are you doing?" I yelled.

"It wasn't right," Tom said.

"Huh?" I stammered. "It was square… I put a level on it myself."

Tom just looked at me and said, "It was perfectly square and plumb, but it wasn't up to my standards. Every time somebody drives past this house, they will look at this porch and know that I put it up." He took his glasses off and gave me a look that could cut steel. "And if it's not what I want it to be, I don't want it attached to my name."

I remember when I was a service tech at a Computerland store. Every time I took the case off of an original Apple Macintosh, I'd look at the inside of it. Right there on the inside of the plastic case was the signature of every person that assembled that Mac.

The developers, testers, and documentation folks I worked with had that kind of pride. Eventually, the organization became more Agile. The team inspected and adapted, and we got better.

Soccer turned out OK as well. We had fun, got some exercise, and learned the game. If I do say so myself, my daughter's team had the best halftime snacks.

Every Scrum Master Coaches

Coaching is like a muscle; you use it or you lose it. As I've said before in this book, there is a difference between coaching and teaching and mentoring. There is also a difference between Agile coaching and professional coaching. Agile coaching is where you help folks with their Agile competency and skills. The goal of professional coaching is to help individuals improve performance. It's about personal empowerment and responsibility. The coach assists the "coachee" by changing thinking and behaviors. When Scrum team members are not meeting expectations, they need coaching.

Ask open-ended questions. They will help the person being coached to think and come up with solutions that are believable to them. They own the solution because they came up with it. Because the Scrum Master isn't telling anybody what to do, they are looked at as a peer, rather than as an expert.

The team member knows what to do. It's the coach's job to help him or her discover that this is the case.

Before engaging in a coaching session, the coach needs to make sure that the team member is open to the idea of being coached. I can remember a friend of mine telling me how he went to see his chiropractor and the guy just grabbed him, threw him on the table, and started working on him. The chiropractor surprised him. The bad thing is that my friend had a pen in his shirt pocket, and he never had a chance to remove it. Needless to say, the chiropractor had to buy my friend a new shirt when the pen broke.

It's the same thing with coaching. You don't just want to start coaching folks out of the blue. Explain that you would like to start a coaching relationship with them and if they are open to the idea.

Once the person is open to coaching, set up a meeting. I make it a point to remove all distractions while the coaching session is going on. No laptops or tablets should be in the meeting. I make it a point to take my cell phone and put it face down on the table in front of me. It's a symbolic gesture that shows that the coaching session is the most important thing going on at the moment.

Start the meeting by letting the person talk. At this point, listen. I mean really listen. Remember, the coach is not here to solve the problem. Don't sit there trying to think of something clever to say or how to solve the problem for the person.

Just listen.

Let them get everything off their chest. It's quite a cleansing, refreshing thing to let everything out and to clear the air.

At some point, the coach will start to ask open-ended questions. For example, "What have you tried to do to this point?" One of my favorites is "If this had gone perfectly, what would it look like?" By having the person being coached explain how the perfect scenario would work, you may just get them to blurt out the solution.

At some point in this conversation, the person being coached will have what I call the "million dollar moment." They will start to talk about reasonable ways to handle the situation. As a coach, I grab on to that and do not let go. Asking open-ended questions empowers the team member. When a coach tells someone what to do, there is an air of superiority there. When the coach guides the person to the answer, the coach comes off more as a peer. This creates buy-in on the part of the person being coached. They now feel that they own the solution.

At this point, there should be an agreement on action items and a follow-up meeting scheduled. The coach needs to make sure that the person follows through with the action items. There is also a strong possibility that the action items may lead to more coaching sessions in the future.

Every single person on the Scrum team has talents that can help make the team better. It's up to the coach to unlock those talents. I will say that coaching sessions can get emotional. People may cry. People may voice their frustration.

Frustration isn't bad. I've dealt with all kinds of frustration during my time as a Scrum Master. Frustrated team members, frustrated managers, frustrated Product Owners; I've even been frustrated myself.

I've grown to learn that frustration is part of progress. If you can learn from your frustration, it can be quite productive. To put it bluntly, you have to care about something to get frustrated by it.

Instead of trying to coach around frustration, I've learned to coach through it. Find out what the real cause of the frustration is and coach the person so they come up with a way to address the problem.

So What Does an Agile Coach Do?

If Scrum Masters coach, why does an organization need an Agile coach?

When a football team wins a championship, do they get rid of all the coaches? After all, the team obviously knows how to win. The best players can now coach the team as they play.

No team does that. A football team needs coaching. So do Scrum teams. An Agile coach is committed to creating high-performing teams. Yes, part of that is developing leaders, but that's not the only function of the coach. The Agile coach works to increase the competency of the Scrum Masters, Product Owners, and managers of an organization.

Random Thoughts

I'd like to pass on some nuggets of wisdom that I've learned on my Agile journey. No rhyme or reason to what I'm sharing here. Just stuff that I think is important to pass on.

Why Scrum?

Scrum provides the means to get feedback before the thing you are building is, well, built. Actively involving customers gives them unparalleled visibility into the development process. Customers and stakeholders see key product changes and enhancements in "real time" as they happen. This translates to the right product. Even a clear set of requirements requires the team to make some assumptions, and the requirements usually change before the end of the project. As requirements emerge and change the Scrum, the team has the ability to embrace the change and provide the value that the customers really need. Flexibility is built into the Scrum framework. Change is expected, and Scrum provides the tools to plan for variability and embrace change. The nature of the iteration exposes problems early. Testing is embedded in the

framework, giving the customer a more stable product because defects are found and fixed earlier in the development process.

Don't Do Stupid Stuff

Command-and-control assumes that you need to be told how to do your job. Yes, that's harsh, but it is reality. Scrum assumes that a team will figure it out by experimenting, inspecting, and adapting. The phrase "we've always done it this way" should be removed from your vocabulary.

Question everything and don't be afraid to inspect and adapt. Scrum culture values openness and collaboration. Don't be afraid to call something out If it doesn't make sense.

However, **realize that you might be the stupid one.**

Have you ever been halfway through an argument with somebody and realized that you are wrong but keep on arguing because of ego? Yeah, that's awkward, and I'll admit that I've done it more than once. Be humble enough to understand that you may need to rethink your position.

My brother has always described me as "subtle as a sledgehammer." I can be arrogant, stubborn, loud, rude, opinionated, and inattentive (sometimes all at once). I'm also genuine. I hope that that makes up for the other stuff I listed.

Fail Fast, Learn Fast

Failure is the best teacher in the world. One of the things I encourage teams to do is experiment and fail if necessary. I just want it to happen quickly. A team failing over and over repeatedly isn't healthy and doesn't deliver value to customers. Fail quickly, learn from it, inspect and adapt, and move on. A team that experiments often and learns from their failures is well on the way to being a high-performing team. A team should experiment so that they can learn what can and cannot be done… period. There is nothing like experimentation to help the team understand its capabilities.

Personally, I feel that failures in life are not only inevitable, but a requirement for getting better. Remember the saying on the back of the shirt my brother gave to me: *You have to lose to win.*

Anyone who tells you otherwise is a liar. Everything you do in life will come with some success and some failure. It's what you do with that failure that defines how successful you will be in the future. You simply can't be afraid to fail; and when you do fail, you need to learn and grow. To be clear, I'm not saying that you should go out and challenge an Olympic sprinter to a foot race. I'm not interested in building a culture of failure. What I am saying is we should set goals and attack them like a silverback gorilla who just finished four energy drinks.

Look at it this way: even if the failure is epic, bigger than you ever could have imagined; even if there is nothing salvageable from the failed experiment, you've only blown a Sprint.

If we are not getting better, we are missing the whole point here. How do you know what works and doesn't work if you don't try new things? Celebrate the successes, learn from failures, and become the best that you can possibly be.

Finish All of the Stories in the Sprint Backlog

The team needs to focus on getting all of the stories they committed to in Sprint Planning completed every Sprint. It's vital that a team does not allow stories to slip into the next Sprint. Stories receive velocity points when they are finished and accepted by the Product Owner. Stories that carry over do not receive partial credit. All story points are recognized in the Sprint the story is completed. Allowing stories to slip adversely affects team velocity.

It's more than just velocity. Not finishing stories in a Sprint makes it tempting to fall back into Waterfall-style behaviors. Remember, the idea is to deliver working software every Sprint. Allowing stories to slip will not allow the team to demo the minimally viable product that they committed to delivering. It also creates a domino effect. When a story slips to the next Sprint, there is a good chance that the team will not be able to finish all of the stories in that next Sprint. So the team does not complete all of the stories again. Stories slip again, and again, and again.

Ambitious estimation or over-enthusiastic team commitments are the usual suspects when a story fails to close in its Sprint. The Scrum team should be trying to under-promise and over-deliver when it comes to Sprint commitments. Get stories and tasks refined down as small as possible so that it's easier to undertake the amount of work necessary to get them closed.

Remember, Agile is all about getting better and delivering stuff that customers want faster. It challenges you to do what makes sense—not to follow protocol like a robot, but to do what makes sense. Get better. Most importantly, do what the Scrum team needs you to do to make the team successful. There is no such thing as idle time during a Sprint. If you find yourself idle, you are doing something wrong. Your first priority should always be the work that the team committed to do during the Sprint Planning meeting, but don't get so wrapped up in protocol that you actually slow down. Agile forces you to focus on what you are trying to achieve.

There is also the issue of taking too much time talking about stories and not enough time actually working on them. Perfection is a mortal enemy of a Scrum team. Remember, working software is what we are after. Get something built and into the customer's hands. Instead of arguing about how a piece of functionality should work, get it in the customer's hands and let them tell you

what they like and don't like about it. What good is getting something just right in the team's eyes but dead wrong in the customer's eyes?

Get the functionality to the point where it is good enough for now, get the story closed, get the customer's feedback, and do something with it.

With Power Comes Responsibility

Agile (Scrum) is supposed to bring joy to the team. The team should "do their thing" without somebody controlling their work. The Product Owner and the Scrum Master have no say in how the team delivers value. The team manages itself.

That can be quite a culture shock to a team that is used to having a development manager tell them how he wants things to be built. This is a great deal of freedom—dare I say great power? And with this comes responsibility.

It is the team's responsibility to do the work… to make the decisions by themselves and deliver what they have been asked. The team needs the authority to make the decisions, and the ability to deal with the fallout from them. It doesn't make sense for a Scrum team to have to wait for somebody four levels up the management chain to give them the OK to do something. Scrum requires that those most impacted by the decision make decisions.

I've heard the Agile rituals (stand-ups, Sprint Planning, Retrospectives, and so on) described as distractions that get in the way of "real" development work.

Um… no.

These meetings are not "overhead." They are there to provide the structure and rhythm necessary for work to get done, and for important information and updates to be communicated between team members. Decisions about direction are discussed and commitments are made to each other. The meetings allow the entire team to participate, collaborate, and (dare I say) self-organize.

There should be no waiting for somebody outside the team to make decisions about anything. The team should have the proper authority to do the job.

It is not the Product Owner's responsibility to decide how the team implements a feature. The PO needs to be focused on the bigger picture, prioritizing and building the Backlog. Let's face it, developers want to build things. By prioritizing features, the PO keeps the team "on point" and not distracted by stuff that is nonessential. If the PO is too involved in the team's development work, she isn't refining and prioritizing. If you really feel that these Agile "rituals" are not worthwhile or simply something that management is making you do, then you and your team aren't taking responsibility for what is going on.

The best Scrum Master in the world can't make any Agile ritual worthwhile if the team isn't interested. In reality, the Scrum team holds all of the cards. There are no more development managers to answer to. The team has the power.

And the responsibility…

Time-box Your Work

I don't think I've defined what a time-box is yet. That's funny to me, because that's what enabled me to write this book in the first place. Time-boxing is limiting the amount of time spent on something. Yes, it's that simple. Use time-boxing to ensure that you stay productive. I admit it; I wasn't a fan when I first heard of it. Now, I couldn't live without it. Time-boxing keeps my natural tendency to procrastinate in check. It helps me to better organize my time. For example, I carve out an hour in my day to work on my book. I set a timer and start writing. After sixty minutes, I move on to my next task. There is something about that moving clock that seems to allow me to focus more. It is also a great way to keep the daily stand-up meeting in check. If there is one complaint that I'd say I've heard the most during my time as a Scrum Master, it would be this: "We have too many meetings."

My usual response is that you wouldn't feel that way if the meetings had value, and then I challenge them to create that value. I have, however, seen a strange phenomenon happen. People may hate all of these meetings, but they won't stop talking when they are in them.

Take the daily stand-up meeting. The stand-up is supposed to last no more than 15 minutes. Technically, it's where team members are supposed to make commitments to each other. That's why it's called a stand-up—it's uncomfortable to stand for long periods, so everybody stands up in the meeting, keeping it short.

That's not what was happening. The team was treating it like an old-style Waterfall status meeting. Folks were talking about stuff that should have been in parking lot meetings, or going down rabbit holes.

The solution was to time-box the meeting. The way I implemented the time-box was to use an obnoxious countdown timer.

I simply set the timer for 15 minutes and let it count down. The results were amazing. The visual representation of time ticking away kept everybody in line, and the stand-ups rarely lasted longer than 15 minutes.

Progressive Overload

How does a Scrum team make gradual and constant improvement over time? Did you ever sit back and think about that? How can you really do that? It

can be quite the daunting task. How do you foster a culture of continuous improvement?

This reminds me of the story of Milo of Croton. Milo was a wrestler back in ancient Greece who was basically unbeatable. He won six Olympic laurels, and performed many feats of strength. He's famous for winning wrestling matches and eating (he supposedly ate 20 pounds of meat a day), but he's most famous for the legend of how he trained.

Legend has it that he'd take a newborn calf, lift it, and then rest it on his shoulders. He would then walk around the city of Croton with that calf on his shoulders. He did this every day without fail. As the calf grew, it got heavier and heavier, and eventually Milo was walking around with a full-grown bull on his shoulders.

The key here is that Milo started with a manageable task. I'm sure that as the calf grew it took effort to lift it. The effort triggered growth, and Milo's body adapted to the load. This principle is called progressive overload. You start with a weight that is manageable and over time add a little more each workout.

It's the same thing with a Scrum team.

Like Milo, we need to be constantly looking for ways to get better. The key is to make small changes over time. Inspect what you are doing, adapt by making small changes, and never stop growing.

You don't take the elevator to agility. You take the stairs—one step up at a time.

One of the dangers that any Scrum team faces is complacency—same old, same old. We need to guard against falling into that trap. If we truly are all about continuous improvement, our focus should be this—how do we make everything better? If you don't find value in what you are doing, it's up to you to determine how to fix that situation. Inspect and adapt! The two most dangerous words in the English language are "I know." Never stop trying to get a little better every day.

Be a Thermostat, Not a Thermometer

Have you ever walked into a room and had the conversation that was happening before you arrived stop dead? That happens to me quite a bit. I'm the guy who "drank the Kool Aid." The Agile evangelist. I can live with those labels, but I'd rather be known as a thermostat. A thermometer tells you the temperature. A thermostat controls the temperature.

When it comes to Agile, I really am kind of like an evangelist. I feel like I'm repeating myself over and over again.

Inspect and adapt.

Be transparent.

How can we deliver value faster?

I believe that relentlessly repeating the message will eventually make it stick.

If I hear you complaining about something Agile-related, I might call you on it, coach you, or ask you how you think we can make it better. I change the temperature. As a Scrum Master and coach, I set the temperature in the room, not just notice if it's hot or cold.

Be Fluid

When you are Agile, you are fluid. Fluid means flexible, adaptable, graceful.

As I think about how we have delivered software during my time in Waterfall teams, the word I think of is rigid.

Rigid—we gave the customers what we thought they wanted. Or worse, we delivered to our customers what somebody in management wanted, but not what the customers wanted.

Scrum is not rigid. That means that we need to be able to react to customer input … to be fluid…

It also means that anything even remotely Waterfall has to die. When you apply Waterfall principles in an Agile environment, you are no longer doing Scrum. What you end up with is a Waterfall project broken up into chunks—not Agile. The teams don't react to customer input The customers are not part of the development process.

I'll explain it this way. Let's say that you hang out at the driving range all of your life. You swing the oversized woods and drive golf balls. Your friend invites you over to play a round of golf. When you get there, you tell your friend that while you may be playing on a golf course, you are comfortable with just driving the ball. You'd rather not use the irons or putter. In this case, you aren't playing golf. In reality, you are just messing around.

In the twenty-first century, we can't afford to be messing around. Requirements are changing so rapidly that our customers can't even get a handle on what they are so that they can give us an idea of what they need. It's vital that we collaborate with our customers so that we can address requirements as they emerge. Even if you say, "I'll play by the rule, but I don't like to use a putter," and you just try to get the ball as close to the pin as possible, you still aren't really playing golf… are you?

Your team is not Agile if you just add meetings to what you always did. If you work on design in the first Sprint, coding in the second Sprint, and testing in

the third Sprint, you aren't doing Scrum. If you deliver value in a Sprint, but it's not tested, documented, and installable, you aren't doing Scrum.

You also shouldn't say you are "doing Scrum." You aren't just "doing Scrum." You are shifting to a customer-focused mindset.

Look To Yourself First

In any situation that comes up, whether in a Scrum team, or some other work or personal context, look to yourself first. Be willing to turn the finger you are pointing at somebody else around at yourself. Ultimately, the person that is responsible is you.

An Agile transformation isn't all butterflies, rainbows, and unicorn kisses. When things go south, and they will, part of being a leader is taking responsibility. Nothing stops the "blame game" dead in its tracks like stepping up and letting everybody know that you could have done better. Scrum is a great framework, but it does do a great job of exposing dysfunction. There are plenty of opportunities for improvement.

If a Sprint Planning meeting doesn't go as planned, because the stories were not properly refined, don't let the team blame the Product Owner. I step up and let the team know that the Scrum Master didn't do a good enough job coaching the PO. I let the team know that I will do a better job of paying attention to what stories the PO wants to bring into the Sprint before the next Planning meeting.

If the daily stand-up meeting goes too long, it's because I'm not stressing the time box enough.

Every time a situation comes up, I look to see where I could have done something differently and take responsibility for it. To me, it's part of being a leader and part of being transparent.

Think of it this way. If I decide that I'm going to plant a garden and go through all of the work of picking out plants, digging up the soil, and putting them in the ground, but the plants all die—was it the fault of the plants? Do you blame the person you bought the plants from, or the ground you put them in?

Or do you look at how you cared for the plants? Did you water them enough? Did you plant them in a place that got enough sun? Did you give them enough (or too much) fertilizer?

The servant-leader needs to take responsibility so that they can drive the team to inspect and adapt. Even if that means you have to fall on your sword sometimes.

Lead the Team

There is a difference between a boss and a leader. A Scrum team needs to be self-organizing; I've said that a bunch of times in this book. Effective teams are also self-managing. The team doesn't need a boss. They need a leader. Someone who supports the team.

An effective leader knows the right questions to ask, and when to ask them. The questions a leader asks encourages the team to think through problems instead of telling them what they need to do. A leader facilitates the decision-making process and helps the team come to an agreement.

Being self-managed does not mean that anybody can now do whatever they want to. There is still control, but that that control now lies in trust. No longer is management responsible for all decisions. They now trust the teams to set expectations and meet them. People are now responsible for their own work—which means that they now self-organize, self-manage, and take initiative. It enables the teams to innovate and take risks.

A good leader enables trust. That might just be the most important thing a Scrum Master can do.

Positive, Positive, Positive

Make the conscious decision to be positive. Every day, in every type of situation, a Scrum Master needs to be the person who "sets the tone" in the room. In the end, positivity is a choice. Positivity is contagious, and it can actually change outcomes. In reality, we truly get what we expect. A positive, "glass-half-full" attitude can make a tangible impact.

The magic number or sweet spot is "three to one," as in three positive statements for every negative one. If you want a team of high-performing rock stars, you need to get that ratio closer to five to one.

For me, this is a tough one. My nature is to respond to a negative comment with a dose of more negativity and a healthy side of sarcasm. In fact, I'm a bonafide sarcasm expert. I've been known to use sarcasm as a defense mechanism, an offensive weapon, and a coping mechanism.

So I've taken it upon myself to address this situation with a rubber band I wear around my wrist. Whenever I feel the urge to go negative (or when the sarcasm monster shows up), I pull the rubber band back and… thwack! I remind myself to not allow that to happen. To say something positive.

Develop an Attitude of Gratitude

Thankfulness is something that doesn't come naturally. It's really easy to get wrapped up in our own little world and not stop and take stock of what's

really important. In fact, it seems that the older I get, the more I notice an astounding amount of a lack of self-awareness all around me.

I was out with one of my daughters the other day, running some errands and taking care of some stuff when I noticed something. Actually, I'd classify it as a bit of a mini-epiphany. We were in a store when I noticed that most everyone around me was annoyed about something or another.

I stood there, in the midst of all the chaos, astonished by the miracle that is a brick-and-mortar retailer. Think about it—yes, the aisles are too narrow, the place is always crowded, but you can find anything you want there. From chicken wings to chicken feed. That is really a miracle. Think of what it takes to get one item available at that store—from production to distribution. Then think of the convenience. If I need, I don't know—a t-shirt at 3 a.m.—I can run there and get one.

I'll give you another example. Back in 2006, I was in Germany for two weeks. European coffee was not quite my cup of tea (pardon the pun). I was the happiest man in Europe when I came across an American chain coffee shop.

I rarely go to this particular coffee establishment when I'm at home. but I was thankful when I found it in Germany.

Let me paint you a picture.

You get to the office and pull your car into a parking spot. There is a spring in your step as you walk into the office. You greet everyone with a smile as you make your way to your desk in order to slide your laptop into its docking station.

It takes you 10 minutes to get the stupid holes on the bottom of your laptop to line up with those little nubs on the docking station. You stomp over to the break room to get your morning cup o' joe and you are greeted by three empty coffee pots. While making a pot of coffee, you curse the guy who took the last cup and didn't have the common courtesy to make a new pot.

That right there is probably enough to ruin anybody's day. That happens to me most mornings, but I don't let it affect my attitude. When I'm having a bad day, I find it refreshing to take some time and list all the things I'm thankful for. It puts things in perspective and allows me to focus on how good I really have it.

In fact, I make it a point to sit down and list seven things I'm thankful for at least once a quarter and post it in my cubicle. It serves as a constant reminder that I really have no reason to be grumpy.

Focus on the Big Rocks First

You are trying to fit as many rocks as you can into a mason jar. Logically, you should start with the big rocks, and then add the smaller rocks and pebbles.

The reason is that if you start with the small stuff, you may not have room for the big stones later. The big stones are the most important ones….the ones with significant impact.

What are the big rocks of Scrum?

The Backlog is Everything, So Pay Attention to It

My experience is that the one Agile ceremony that teams seem to want to skip the most often is the Backlog Refinement meeting. This is a huge mistake. The Backlog is the lifeblood of a Scrum team. It should be treated as something precious, because it really is. The Backlog is a prioritized list of "stuff." Specifically, it's the "stuff" that might be included in your product as well as the requirements for these changes. It needs to be the single source of requested changes, and it needs to be visible to anyone.

Simple enough, right?

Here is the important part: The Backlog needs to be a living artifact. Items in the Backlog should evolve and possibly change as the team learns more about how customers will use the required functionality and the environment in which it will operate. If a team and Product Owner are not regularly refining stories in the Backlog, you will not produce valuable features for your customers—it's that simple.

Be Absolutely Relentless About Feedback

Feedback is the reason why we work in short iterations. It's the reason why we do Sprint review meetings. The lack of feedback is why the Waterfall methodology fell out of favor with so many teams. Feedback is important, so we need to go after it relentlessly. It's not good enough to just have a Sprint Review meeting. You need to engage you customers.

You do have customers in your Sprint Review meetings… right?

You need to have as many customers as possible giving you feedback. This means you need to do whatever possible to get the feedback your team craves. If customers don't offer up what you are looking for, ask for it.

It's Inspect and Adapt, not Complain and Refrain

Agile should be like nirvana to anybody involved with it. Why? Inspect and adapt. You should be looking, or at least be thinking, about ways to make everything and anything better… constantly. When I hear folks complain, I always wonder why. You have the unique opportunity to do something about it. I understand that I'm known as the "glass-half-full" guy. A positive attitude

is a must, especially in an Agile environment. If you don't like it, figure out how to make it better.

Embrace DoD and DoR

As I detailed in Chapter 4, before taking off, a pilot does something called a pre-flight checklist. This is done so that the pilot has no doubt that the plane is air-worthy. No pilot would go down the runway without doing a pre-flight check. It's the same with the Definition of Done (DoD) and Definition of Ready (DoR).

How do you know when a story is ready to be pulled into a Sprint? You know by the DoR. By ready, we mean *ready* ready, not kind of ready, or sort of ready. The DoR is a collection of all the stuff necessary for a user story to be developed. The DoR can change and evolve through the release, but there should be only one that is used for all user stories. If the DoR is properly defined, everybody on the team should be aware of when a story can be pulled into any Sprint.

The DoD is what the team should be working toward. It gives the team a great way to report progress on a feature. Think of it as a checklist that will validate not only that all of the tasks are accounted for, but that you didn't forget to do anything in the pressure of a time-boxed iteration.

The DoD and DoR need to be living documents that grow and change as the team reaches new levels of Agile maturity. The DoD and DoR allow a team to never pull anything into a Sprint that is not ready, and never let anything out of a Sprint that is not done.

Make Sure You are Talking to the Right People

When we invite customers to our Sprint review meetings, we want folks who are, as I've said before, on the glass. In other words, people who are actually using our products. The people who are sitting behind the computer, or tablet, or mobile device and using the software that we produce. On the glass.

Doesn't it make sense to get feedback from the person who actually uses the product?

To me it's a no-brainer, but are we doing it? So many times I see management, or the VP of purchasing, or somebody waaaay up the chain of command interacting with us.

Think of it this way… If I opened up a lemonade stand, would you ask the kids who are drinking the lemonade what it tastes like or would you ask the parents who bought the lemonade for their kids?

I'm sure that the parents would say stuff like:

"You are charging too much."

"I don't like the way your booth looks."

Notice, nothing about what the drink you are selling tastes like. In reality, there is no difference with software. We are trying to make the people who use our software more like fans than customers. But don't we want to focus on those who control the purse strings? You know, the guy who makes the purchasing agreement?

Let me answer that question with a question. Do you think that the folks in the data center (on the glass) are happy when you take away the software that they like to use? From what I've seen at customer sites, when the users aren't happy, nobody is happy.

There is nothing wrong with including people like purchasing agents and vice presidents in Sprint demos, but make sure that the people who use the software are there as well.

Final Thoughts

I'd like to close this book with a story and a thought. My journey to Agile has been, well, *unique,* to say the least. I'd like to share a bit more insight before I close things out.

Never Forget It's All About the Customer

My friend Ken McGaffic runs a festival called Old Fashioned Christmas in the Woods each year in October. I help him out, usually selling admission tickets. I'm sure Ken doesn't mind that I look like Santa Claus. I have to admit, that comes in handy at a Christmas festival. Me, I'm a people person, so I'm in my element. I get to talk to people and make new friends. I also get to see firsthand what it looks like to have customers who act more like fans.

Thousands of folks make their way to the Shaker Woods festival grounds in Ohio to visit 215 craft booths (and/or 25 food vendors), looking to find that special something for the holidays.

The most amazing thing to me is what I find out talking to people. I'm the kind of person who will talk to anybody. As I said before, I'm a people person. One of my responsibilities at the festival is to "guard the wire" before the show opens. What I do is run caution tape through the part of the festival where people walk in and keep the customers at bay until the vendors are ready to start the day. Once again, looking like Santa comes in handy. My size also works out well in this situation.

While standing at the wire, I talk to folks. It's a great way to pass the time for both the customers and me. As we chatted, I kept getting comments like this:

"I look forward to this every year. Over the years, we've made a family event out of it."

"This is the best craft show on the East Coast."

What does a craft show have to do with Agile? In a word, everything. The Shaker Woods fairgrounds are literally in the middle of nowhere. One of those "make a right at the third cornfield" kind of spots, yet people show up from over 30 states every year. As part of Scrum, we are trying to get customers more involved in the development process. We ask them to join us at the end of our Sprints and talk about what we are doing—and get feedback on what they like and don't like. The idea is that if we give the customers what they want and what they need, or create stuff that they look at and say, "I gotta have that!" then the sales process will take care of itself. In other words, we won't have to sell anybody on our products. Our customers will be clamoring to get their hands on them. That's what I want. Customers who will stand out in the woods on a chilly October morning because they want what is being sold.

So, how did my friend Ken create the same type of customers we are looking for?

He tailored the show to his target customers. He feels that women over the age of 25 like the quaint, outdoor setting. As he put it, "They like to take a walk in the woods without getting mud on their shoes." He knows what his customers want and is trying to delight them.

The crafts that are sold at the festival are all handmade by the crafters. The quality is very high. To get a place in the festival, a crafter must not only have handmade crafts—they are required to be in costume and to do demonstrations of how they make their crafts during the show.

We are both after the same thing—customers that love what we produce and keep coming back for more. We both pretty much are trying to do it in the same way. Listen to customers, give them what they want, and make sure it's of high quality.

Don't ever forget it's about the customer. When we think that we are the "voice of the customer," when we value writing code over delivering value, we no longer are serving the best interest of the customer. They will go somewhere else—to somebody who is serving them.

When is the last time a customer thanked you? One of the things that I enjoy the most about my time at Christmas in the Woods is that as people are leaving the festival, they thank me. Not a half-hearted thank you as they brush past me on their way to their car, but an honest-to-goodness thank you for helping make the festival what it is.

They Are People

A Scrum team is a collection of people. Never forget that. Respect the "humanness" of the folks on the team. Actually, respect the human-ness of everybody you meet. My experience is that most people are not unreasonable; it's the stress of life that makes them that way. During the writing of this book, my father died, my mother-in-law was diagnosed with breast cancer, and a good friend of mine with whom I played music for years also passed away unexpectedly. All of that caused even me to be a little less positive than usual.

I really believe that nobody gets up in the morning and plots out how he or she is going to be difficult while taking a shower. The person who has an attitude is probably under more stress than you can imagine for some reason or another. It may not even be something at work that is working on them.

Scrum Mastering and coaching is more about people skills than the Agile Manifesto. The framework is important, but the people are more important. One of the things I ask people all the time is if they are happy. A Scrum Master is the responsible for the happiness of the team he or she leads.

I'll leave you with this. If you show people that you truly care about their well-being, they reflect that. Not only back to you but to those around them.

That makes Scrum work.

I

Index

A

Acceptance criteria shows you what success looks like, 112

Acceptance test driven development (ATDD), 167

A coach sees value in everybody, 143

Adversarial relationship with development, 43

Agile coach, 98, 128, 141–143, 154, 171, 191, 194, 196

A powerlifting belt, 90

Architecture, 24, 113, 130–131, 164

A shoot, 67

Assigning story points, 118

Assist the scrum master with impediments, 144

Attitude of gratitude, 204–205

Authority, 39, 41, 42, 59, 104, 119, 136, 138, 144, 145, 182, 184, 185, 193, 199

Automated build, 166

Availability to the team, 134–135

Average velocity, 83, 116–118, 172, 178

Awareness, 137, 138

A work, 67

B

Back burnered work, 105

Backlog, 18, 21, 27–30, 32, 34, 39–43, 60, 78–85, 87, 91, 98, 107, 108, 117, 119, 120, 123, 128, 130–131, 135, 136, 145, 158, 160, 164, 173, 174, 176, 178–179, 184–186, 193, 198, 199, 206

Backlog defined, 28–30

Backlog refinement, 89, 110–111, 114, 174

Backlog refinement meeting, 206

Beaconing, 150

Big rocks, 205–206

Bombing out, 168

Bouncer, 154

Bricklayer, 193

Buckets, 92, 173, 174

Build process, 166

Build rapport, 140

Build something the customer will use, 78

Burn-down chart, 85–87, 120, 152, 178, 185

Burnout, 14, 140, 170

C

Cancel a sprint, 185

C and C team builder, 139

Capacity planning, 58

Career advancement, 158

Celebrate, 139, 158, 198

Change is scary, 122

Changing team personnel, 182

D. McKenna, *The Art of Scrum*, DOI 10.1007/978-1-4842-2277-5

Coaching, 21, 24, 35–36, 50, 54, 136, 138, 141–144, 149, 150, 152–154, 158, 168, 191, 194–196, 203, 210

Coaching is about bringing greatness out of people, 150

Coaching session, 144, 195

Coaching soccer, 189

Collaboration, 17, 45, 55–58, 137, 139, 155, 177, 197

Collocated teams, 55–57

Comfort zone, 129, 141–144

Command and control, 4, 38, 87, 144, 151, 152, 157, 193, 197

Command and control is about lack of trust, 151

Commitment, 43, 57, 58, 88, 125–126, 136, 160, 176, 179–180, 184, 198–200

Commitment by the team, 34

Communication, 17, 21, 22, 50, 53, 55, 58, 100, 152, 156, 184–185

Compile, 22, 104, 125, 166

Complacency, 201

Complaining about meetings, 126

Complexity, 18, 23, 28, 45, 113–114, 137–138, 165, 173

Component teams, 59

Congratulate, 139

Continually remind them of the good things, 151

Continuous integration, 103, 165–166

Cost of waiting, 173

Countdown timer, 200

Courage, 43

Creating a DOD with sticky notes, 163

Creating tasks, 176

Creating the DoD, 104

Cross functional, 30, 42–44, 57, 61, 92, 99, 157–158

Cross functional, blur the lines, 43

Culture of continuous improvement, 5, 38, 201

Customer collaboration over contract negotiation, 17

D

Daily standup, 51, 53, 59, 60, 72, 87–88, 116, 119, 120, 124–126, 145, 152, 182, 190, 193, 200, 203

Dead garden, 199

Decompose stories into tasks, 91, 102, 176, 178

Defects, 10, 11, 13, 28, 58, 74, 80–81, 83, 116–118, 145, 179, 187, 197

Defect trends, 187

Define a DoD, 192

Definite, 171, 173, 185

Definition of Done (DoD), 32–33, 43, 65, 70–76, 80, 103–106, 163–164, 170, 192–193, 207

Definition of ready (DoR), 72, 75–77, 85, 106, 107, 110–111, 163, 192–193, 207

Demo, 33, 74, 80, 91, 103, 109–110, 127, 134, 135, 145, 160, 165, 183, 198

Describe agile in one sentence, 5

Describe a team you admire, 99

Design and architecture, 164

Design phase, 9, 10

Desired outcomes, acceptance criteria, 41

Developers, 4, 10–13, 16, 17, 18, 20–23, 39, 40, 42, 44, 45, 80, 98, 104, 110, 131, 154–155, 157–158, 170, 177, 191, 194, 199

Development manager, 16, 21, 24, 33, 36, 42, 48, 83, 87, 98, 130, 152, 199, 200

Distributed teams, 57–58

DoD. *See* Definition of Done (DoD)

DOD is a living document, 73, 164, 207

DOD is separate from acceptance criteria, 65

Doing agile, 49, 51, 138
Dominate the conversation, 156
Don't boil the ocean, 94, 192
Don't do stupid stuff, 197
Don't out kick your coverage, 177
Don't paralyze the team, 57
Don't say how you want it. Say what you want., 109
DoR. *See* Definition of ready (DoR)
Driving range, 202

E

Early testing removes risk, 44
Educating the organization, 160–161
Effective leader, 204
Elephant, 190, 191
Emergent architecture, 131, 164
Emphasizing what the user wants, 135
Empower the PO to do their job, 136
Enforcing rules, 100
Epic defined, 28
Epics, 28, 82–83, 130, 170–173
Esprit de corps, 57
Estimating, 14, 75, 114, 178
Everybody is valuable, 150
Everything magnifies., 61
Executive buy in, 193

F

Face to face communication, 21, 53, 56–57, 184
Facilitating, 33, 45, 52–53, 58, 67, 70, 135, 156–159, 183, 184, 204
Facilitating meetings, 126, 135, 141
Facilitator, 52–53, 58, 70, 159
Factory issues, 125
Fail fast learn fast, 197–198
Familiarity with development work, 115
Familiarity with people and personalities, 154
Feature teams, 58–59
Feedback, 18–20, 23, 27, 31, 33–34, 78, 79, 83, 118, 127, 151, 173, 177, 183, 196, 199, 206–207, 209
Fibonacci sequence, 114–115
Filter, 160
Finish the stories in the sprint backlog, 198–199
Fist of five, 67, 74, 94, 101, 105
Fluid, 83, 202–203
Focus, 17–22, 24, 28, 30, 36, 39, 41–43, 45, 51, 53, 55, 65, 71, 79, 88, 94, 101, 110, 112–113, 119, 120, 122, 127, 129–130, 135, 151–153, 160, 163, 167, 179, 183, 193, 198, 200–201, 205–206, 208
Focus on the organization, 142
Focus on the outcome, 65, 112–113
Frustration, 155, 163, 195, 196
Functional requirements, 28, 71, 78

G

Get to building stuff sooner, 164
Give them a job, 24
Glass half full, 51, 159, 204, 206
Goals, 6–8, 17, 18, 20, 24, 30, 34, 41–43, 52, 53, 57, 59, 82, 89, 94, 101, 109, 113, 165, 179, 190, 194, 197
Going dark, 134
Golf, 202
Good listening, 153
Grilled cheese, 137
Group discussion, 56
Guide everybody to positive steps, 155

H

Hardware is not expensive, 157
Help by facilitating, 135

Hitting depth in the squat, 150
Hockey stick scrum, 177
How does the team work together, 100–101
How to build a DOD, 104
Humility, 197

I

Impediments, 43, 48, 53, 60, 88, 118–120, 144–145, 152, 155, 181–182, 184
Implementation phase, 10
Incorrect estimates, 178–179
Individuals and interactions over processes and tools, 15–16
Information radiators, 35, 152, 182
Inspect and adapt, 24, 38, 57, 92, 128, 129, 138, 159, 185–186, 193–194, 197, 201–203, 206–207
In the loop, 145
INVEST scale, 76
Iron triangle, 169–170, 185
Is this really an impediment?, 182
It's about the customer, 209

J

Janitor, 46, 152

K

Keep the conversations private, 141
Keep the team together, 99
Keep user stories as small as possible, 112

L

Left on the field, 150
Lemonade stand, 79, 207
Let conflict play out, 154
Lights out, 166
Limiting WIP, 121–122
Listening, 49, 50, 78, 88, 101, 139, 152–153
Look at the story from the customer's perspective, 78, 89
Losing expertise, 158
Lumberjack, 191–192

M

Macintosh, 194
Maintenance phase, 11–14
Managers and scrum masters should be on the same page, 144
Managing WIP is a sign of maturity, 122
Manual build, 166
Marathon training, 82
Mastery, 137, 138
Mentoring, 45, 144, 153, 194
Merging code changes, 165
Metrics, 186
Million dollar moment, 55, 195
Milo of Croton, 201
Minimally viable product, 19, 20, 33, 165, 198
Multiple team backlog, 184
Multitasking kills conversation, 152

N

Non definite, 171, 185
Non-functional requirements, 28, 71, 78
Number one fan and cheerleader, 139

O

Obese backlog, 83, 84
Old Fashioned Christmas in the Woods, 208
One wring able neck, 39, 119, 134, 136
Only the first three sprints are populated, 172, 173
On the glass, 78–79, 145, 207–208
Open ended questions, 194–195
Openness, 43, 151, 197
Organized around features, 59
Overemphasize the DOD, 74
Overhead, 183, 199

P

Pair programming, 157
Parking lot, 88, 119, 139, 154, 159, 200
Part time teams, 58
Perception, 89
Perfection, 198
Personas, 79, 109–110, 167
Picking the team, 98
Pick your opener, 168
Planning for variability, 175
Play an octave lower, 136–137
Positivity, 51, 92, 118, 159, 204
Post the DOD in a public place, 71
PowerPoint, 183
Pride, 88, 100, 184, 193–194
Principle 1: Our highest priority is to satisfy the customer early and continuous delivery of valuable software, 18–19
Principle 2 : Early and continuous delivery is the mechanism we will use to delight our customers, 19–20
Principle 3: Welcome changing requirements even late in development, 20
Principle 4: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale, 20–21
Principle 5: Business people and developers must work together daily throughout the project, 21
Principle 6: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done, 21–22
Principle 7: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation, 22
Principle 8: Working software is the primary measure of progress, 22–23
Principle 9: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely, 23
Principle 10: Continuous attention to technical excellence and good design enhances agility, 23
Principle 11: Simplicity-the art of maximizing the amount of work not done-is essential, 24
Principle 12: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts, 24–25
Prioritize the backlog, 41, 135
Prioritizing the backlog, 29, 78, 135, 173, 199
Product owner as customer advocate, 39
Product owner plays customer, 89
Professional coach, 149–150, 194
Proficiency, 133, 137–138
Progressive overload, 200–201
Project manager, 144–145
Protect the team from outside interference, 160
Pulling additional work into an iteration, 123–124
Put action items in the backlog, 128

Q

QA no longer victims, 44
Quicksand, 181–182

R

Recap of Scrum, 34
Reducing scope, 185
Refactoring, 165, 171
Release burn-up, 87, 185–186
Release burn up chart, 87, 152
Release plan, 17, 18, 75, 82, 133–134, 167–173, 180, 183, 185–186
Remove distractions, 152, 195, 199

Remove impediments, 48, 119, 155
Removing impediments, 144, 182
Reprioritize the backlog often, 135
Requirements, 6, 10, 12–14, 17–20, 24, 28–30, 34, 39–42, 59, 77–78, 89, 113, 125, 130, 136, 145, 164, 165, 173, 196–197, 202, 206
Requirements change rapidly, 12, 17, 113, 202
Requirements phase, 8–10, 12
Respect, 43, 91, 101, 141, 152, 210
Responding to change over following a plan, 17–18
Responsibility, 32, 39, 44, 78, 88, 101, 119, 125, 135, 194, 199–200, 203
Retrospectives, 24, 33, 34, 38, 92–94, 128, 141, 144, 156, 164, 174, 185, 199
Return on investment (ROI), 27, 39, 40, 145, 171, 173
Roll call, 183

S

Sailboat, 93
Satisfy the DoD, 32, 33, 71, 75, 103, 105, 110, 192
Scaling scrum, 58–59, 61
Scope creep, 185
Scrum as a garage band, 136–137
Scrum master, 21, 27, 30, 33–35, 38, 43, 45, 46, 48–49, 51–54, 58–59, 64, 86–89, 91–93, 97–187, 189–191, 194–196, 199–200, 202–204, 210
Scrum master community of practice, 141
Scrum master has no authority, 138, 144, 184
Scrum masters do not write code, 159
Scrum of scrums, 59–61, 184
Scrum of scrums meeting, 59–60, 184
Scrum roles, 34–54, 146
Scrum teams are people, 210
Scrum values, 43
Self directed, 99, 125
Sell, 20, 79, 87, 118, 208–209
Servant Leader, 48, 119, 130, 138, 146, 149, 152–153, 158, 203
Setting capacity, 102
Setting the stage, 99–100
Shackled to the old way of doing things, 190
Sharpen your axe, 192
Show off, 182–183
Simulation with plastic blocks, 134
Sizing have and estimating are two different things, 178
Source code, 167
Spike, 124, 131, 190
Splitting stories, 115
Sponsor, 22–23, 145, 170, 186
Sprint burn-down, 86, 120, 152, 178–179, 185–186
Sprint goal, 30, 43, 89, 91, 102, 119–120, 123–124, 160, 176, 182, 185
Sprint planning, 30, 85, 86, 89–92, 102–103, 124, 134, 138, 160, 173–176, 178, 183–184, 198–199, 203
Sprint review, 31, 33–34, 77, 113, 174, 182–183, 206–207
Sprint reviews and demos, 127, 135
Stakeholders, 13, 30, 31, 33–35, 38–43, 65, 74, 77, 80, 83, 105, 108, 113, 115, 116, 127, 134–135, 144–145, 151, 167, 171, 183, 196
Standup is not a status meeting., 31, 126
Story points, 75, 76, 81, 83, 87, 90, 111, 114–118, 124, 156, 171, 178–179, 198
Story title, 108
Story writing, 107
Strong disagreements, 154
Studio wrestling, 63
Support, 17, 18, 21, 23, 30, 35, 39, 40, 52, 67, 68, 80, 128, 141–142, 144–146, 155, 158, 160, 175, 185, 190, 204
Support the PO, 134
Sustaining engineering, 11, 48, 58

T

Take action before a problem arises, 179
Tasks, 32–33, 73, 76, 83, 85, 86, 89, 91–92, 102–105, 112, 115, 173, 176–179, 185, 198, 207
Teaching, 49, 134, 153, 194
Team authority, 42
Team celebrating, 157
Team involvement, 125
Team mom, 54
Team rules (team agreement), 100–101
Team tools, 164
Test automation tasks, 103
Test driven development, 45, 103, 157, 166–167, 177, 190
The agile cup of coffee, 32
The agile manifesto, 14–18, 25, 34, 49, 51, 115, 210
The ask, 89
The Cheetah, 4
The fourth industrial revolution, 3, 4
The more agile antelope, 4
The power of coaching, 143
The product owner, 28–30, 34–35, 38–43, 54, 71, 77, 78, 80, 83, 85, 89–92, 98, 105, 107, 111, 113, 119–120, 125, 127, 133–136, 158, 160, 164, 171, 173, 176, 178–179, 182, 185, 198–199, 203, 146152
The product owner and the backlog, 41–42
The product owner does not write all of the stories, 135
The quality assurance engineer, 10, 13, 43–45
There is power in positivity, 159
The scrum team defined, 42
The sprint, 20, 30–35, 38, 43–45, 58, 65, 76, 77, 81, 83, 85–86, 89, 91–94, 102, 103, 110, 113, 117–121, 123–124, 127–128, 134–135, 139, 145, 160, 164, 173, 175–179, 182–183, 185–186, 198–199, 203
The team is the heart of scrum, 57
The three pillars of scrum, 35–36
The twelve principles, 18–25
The value of the daily stand-up, 126
Three questions, 34, 88, 119
Timebox, 31, 45, 126, 128, 154–155, 159, 193, 200, 203, 207
Timebox (definition), 200
Time, cost, and scope, 169
Tools, 15–16, 57, 59, 73–75, 109–110, 153, 164, 196
Transparency, 35, 38, 43, 53, 58–59, 87, 103, 151, 171, 184
Trunk, 165

U

Understand scope, 182
Untapped resource, 145
User stories defined, 28
User story must stand on it's own, 80
Use technology to get the team communicating, 57
Us versus them, 13, 57

V

Velocity, 81–83, 87, 111, 116–118, 122, 136, 145, 158, 171–172, 178, 182, 185, 198
Verification phase, 10–11, 13

W, X

Waterfall, defined, 6, 7
Weapon of mass destruction (WMD), 154
What does done mean?, 65, 104
What does it mean to be done, 104
What is a defect, 80
What is consensus, 52, 66–73
What is technical debt, 116, 117

Who attends?, 59
Who writes user stories, 41
Why scrum?, 130, 196–197
Why we don't story point defects or debt, 116
Wiggle room, 173, 185
Wiki, 91, 186
Within fifty feet of each other, 56
Working software over comprehensive documentation, 16–17
Work in progress (WIP), 121, 122, 184
Work-life balance, 23, 140
Wrestling lingo, 87

Y

Yeast, 142
Yellow card, 156
You have to lose to win, 121–122, 184
"You need to lose to win", 37–38

Z

Zero defects, 17, 44, 73, 74, 80, 83, 117, 180, 187