CHAPTER 1

Introduction to Agile

CONTENTS

Agile User Experience Design. http://dx.doi.org/10.1016/B978-0-12-415953-2.00001-7

INTRODUCTION

To understand what it means to practice an Agile form of user-centered design, it is important to have a sense of what exactly *Agile* means and where the term came from. Since the Agile methodology has a deep, rich history and is continually evolving, it has become the subject of many books, blogs, white papers, conference presentations, and websites, all of which have their own take on the value system and its methods. This chapter touches on only the most common terms and concepts and those that might be most relevant to a user experience practitioner. I encourage everyone to spend some time exploring the many resources that are available to get a deeper understanding of the philosophy and the various methods for applying it that have grown up along the way. It is also important to recognize that there is no one single right way to implement Agile design. At its core, "Agile" is a set of values to use as compass to guide a team through the production of software. Whatever process or tools are used and how they are applied are secondary to the overall goals of empowering a highly functional team as it builds great software to the delight of its end users.

Agile is a term that grew out of efforts in the 1990s to find a better development method for producing software. Traditional methods, such as the waterfall method, were starting to be recognized as a bit unwieldy. Consumers were expecting more of their software in terms of quality and functionality, and production cycles needed to change in order to create a product that would satisfy end users. Additionally, production cycles needed to be able to adapt and accommodate the reality of shifting requirements. Waterfall development makes it especially challenging for teams to respond to issues, mostly because it is inherently unable to discover serious problems with the design, architecture, or the code itself early in the cycle and can really identify them only when it is too late for a correction. Not knowing what problems might exist until the end of the release cycle results in a lower-quality product or longer release cycle. Traditional methods are also prone to creating silos, where product managers throw requirements "over the wall" to designers, who then throw their design specifications over the wall to development teams, who throw their code over to

a quality team, that eventually authorizes the release of a product. Due to the limited interaction and communication among these teams during the production of each deliverable, each team is playing a game of telephone and putting its own interpretation on the requirements or the specifications. As in the telephone game, the end result very rarely matches the original intention.

Development teams began experimenting with new techniques like Extreme Programming, Adaptive Programming, Scrum, and other methods to find a better way to produce high-quality software and meet demands without requiring developers to write code 24 hours a day. In 2001, practitioners of a variety of these philosophies got together in Utah and created the Agile Manifesto. The manifesto, and its accompanying 12 principles, captures the spirit of what all these methods were trying to achieve and is an important starting point for an organization that is consider adopting Agile practices. Reading the values expressed in the Agile Manifesto is always a good way to check on whether or not you are on track with your own Agile implementation. While the methods, techniques, and terminology have evolved since 2001, the core values of Agile have not. The manifesto eloquently states:

> "We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:
> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
> That is, while there is value in the items on
> the right, we value the items on the left more"
>
> **The Manifesto for Agile Software Development, agilemanifesto.org**

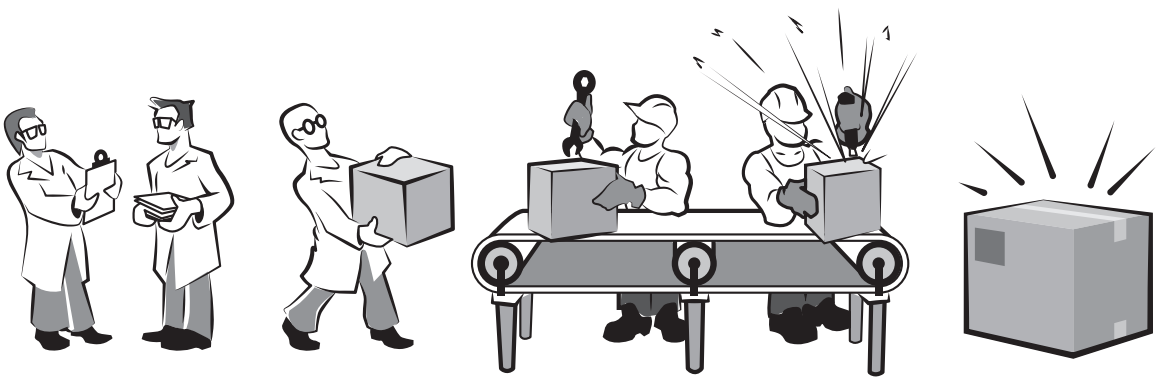

FIGURE 1.1
The agile process.

AGILE VALUES + UX

The manifesto provides the best guidance for and is the most succinct expression of the values that should drive an Agile organization. A description of the principles behind the Agile Manifesto is provided on the website, and it gives additional insight into the manifesto, but the heart of the matter is very well expressed in the manifesto itself. While referencing something that was written so long ago might feel old fashioned, I find that these values are just as relevant now as when they were first created. In software years, 2001 was a very long time ago; and while it was not quite the Dark Ages, it might well be analogous to the Atomic Age. After all, 2001 was the year that many overvalued dotcoms went under, the first-generation iPods were introduced, Microsoft released XP, and several years ahead of the introduction of the ubiquitous Facebook. UX (user experience) practitioners are working today who have no memory of that time. With all the focus on Agile in recent years and the birth of so many variations on the theme, it is fair to assume that the movement has evolved beyond its original manifesto. After all, how many software concepts still hold water over a decade later? That assumption is false. All the children of the original Agile movement still share the core values expressed in the Agile Manifesto and really just use different tactics to create an environment that embodies the spirit of the original statement. All these years later, software production teams still struggle with processes that often focus on the milestone deliverables instead of keeping their eyes on the shipping product. Rarely is a release cycle not subject to changing needs from stakeholders, but rarer still is the process that can support responding to such a change—unless you are working in an Agile environment. The intention of the Agile Manifesto is to define a value system that allows for the creation of a culture that can respond to these situations, while recognizing the value of the individual team member, and produce good software. While the Agile Manifesto lays out the core values of an Agile environment, many techniques can build a process and a framework to support those values. Examining these values through a UX lens can show that there is a natural relationship between the two.

Unfortunately, at no time have UX professionals gotten together and hashed out a manifesto of UX values and principles that came to be the standard to which we all hold ourselves. In fact, despite definitions of the process for user-centered design, there is no formal expression of its value system, beyond the idea that we want to keep the user's needs central to the design process. Things start to get even more vague regarding UX principles. It is not that no UX principles are written down, but that different UX principles are written down by many people. Quite a few people have their own spin on UX design principles, including Microsoft, which has defined the UX principles for Windows and shared these principles on its website (see http://msdn.microsoft.com/en-us/library/windows/desktop/dd834141.aspx). Most of the definitions of UX

principles tend to mention the mission of keeping the user involved in the process, then identify additional guidelines for best supporting the user. Rather than picking a favorite to use for a comparison to the Agile principles, since I have yet to see a set that did not offer some interesting food for thought, or creating yet another set of UX principles, we will review the Agile principles and values and explore the way they fit with and support the type of activities in which UX practitioners tend to engage.

Individuals and interactions over processes and tools

Despite being the first value expressed in the Agile manifesto, it can often be the first one that is forgotten as teams explore the different tools used to manage tasks, user stories, and generate burn-down charts. Excitement over various ceremonies, like daily scrums, backlog grooming, planning poker, and retrospectives, and a desire to properly execute those meetings can often distract the teams from focusing on the people and the communication these things are intended to support and inform. Add to that the fun of exploring and using new software tools to support the process, and the team's attention can be easily aimed in the wrong direction. If a daily standup meeting needs to run for 20 minutes while the team becomes accustomed to the brevity of the status reports, that can be okay as long as the teams are learning how to give brief status reports and they are improving their ability to share only the relevant information. The point of 15 minute meetings is less about that specific amount of time, although it is a very good target, and more about providing the least amount and most relevant information in a meeting short enough to not create a burden for the team to attend. If the team is consistently allowing such meetings to run for 30 or 45 minutes, then it has a problem that needs to be resolved. Similarly, if planning poker is taking too long, does not feel productive, or is causing team members to become disengaged, then it is time to look at whether planning poker is the best technique for the team to use, or if there might be a better way to estimate effort. While it is fun to play with a set of cards, the event is just one way to get the team to participate in effort estimations, and it is not the only way to do that. If the team is several sprints into the cycle and members put more time and energy into the task management tools than they get out of them, the team needs to talk about a different way to handle and track tasks. Even if it would be too disruptive to switch process managment tools during the release, a healthy discussion about the tool will identify problems and potential solutions and help determine if something different needs to be used in the future. It is a critical part of the Agile method to refine the process throughout rather that pick an approach and stick to it no matter what. All these techniques are meant to be the means to an end; and if they are not moving the team toward increased and improved communication, then it is

time to revisit the Agile Manifesto and think about how to actually be more Agile.

From a user experience perspective, the idea of treating the process with the same iterative design techniques used on the product should feel very comfortable. We ask customers for their feedback on the product and refine it accordingly; the product team is the consumer of the process and should be able to provide input into its design. Additionally, designers and usability specialists are used to thinking about the individual end user rather than a set of features; this lends itself well to a process built around the idea of user stories. Improving the interaction between the end user and the system is always foremost in our thoughts. After all, some of us have even had "interaction designer" as part of our job title. Going Agile means taking the same perspective that we have on our users and applying it to our teammates and our work. Some UX folks do this naturally and focus on relationship as a matter of course. Since UX teams are rarely able to realize their designs and implement them for the shipping product on our own, we rely on the development team to do so; and it is always in our best interest to build healthy relationships with the team and other functional groups with which we work. However, it also happens that there are team who perhaps have fallen into a more "us vs. them" mentality with their colleagues from other functional areas. Practicing Agile is a great opportunity to let go of any divisive mentalities and focus on team building. The good news is that this particular Agile value gives everyone on the team permission or instruction to focus on communication and the individual team members rather than on the process. Recognizing that each team member brings a unique set of skills to the team is also a change from traditional processes, where the developers often are seen as interchangeable code producers. Valuing individual skills provides an opportunity to recognize what the user experience team members bring to the table. It also allows the process to be tailored to the individuals that constitute the team.

Working software over comprehensive documentation

Obviously, the highest priority in any production cycle should be the thing that is created, sold, and generates revenue. However, if a functional area is rated and rewarded more for the artifacts it produces than for the shipping software, this priority can easily get lost. People want to do the right thing, but they also want to be recognized for their work; and if such recognition is given to only a very specific piece work or deliverable, then that is where their energy and attention will go. Unfortunately, many traditional methods also require so much documentation that they end up creating environments where it is more important to generate the items that support the process than to produce the software itself. Obviously, the well-intentioned thought is that the whole of the

software is made better by the increased quality of its parts, but the myopic focus on the production of the bits and pieces can create silos that inhibit the communication that might bring this hope to fruition. The idea of putting the attention on the development of the software over generation of the supporting documentation can represent an interesting challenge for UX practitioners who want to take this value to heart. While we certainly contribute to the product, in most cases we usually are not responsible for building it, but we can and do create plenty of supporting artifacts.

In many ways, this principle is asking us to let go of our most visible and tangible contribution to the release. But, if we can make peace with that idea and recognize that the product is more important than the detailed internal deliverables, we can also see that this could positively affect our relationship the other functional areas, as it can reduce barriers to communication and put our efforts toward attending to what gets implemented rather than being satisfied with achieving a good design. Designers, despite our best efforts to remain open to all solutions, often get focused or attached to a particular design solution and wrapped up in the process of creating and documenting that design. However, if little or none of that elegant solution makes it in to the product, was it a good investment for the company or the designer and did it do anything to improve the end user's experience? This does not mean that a UX person should live in a constant state of compromise and cave in on important issues just to get some of the design elements implemented. But, it does mean reminding ourselves that the design unto itself is not the end goal; instead, the design of the shipping product should be the focus of all efforts.

This more holistic approach requires that we to take a closer look at the way we work with our colleagues to see if we are doing everything we can to educate all the functional areas about the value of a good user experience and exactly why a given design solution is important to the customer. If our teammates truly understand the value of a particular design element and why it is of benefit to the end user, they will not be as quick to make trade-offs that compromise its value and may make more effort to work with the designer to find an equitable solution when any kind of trade-offs need to be made. It would also support this core value to emphasize relationship building with other functional areas to improve the dialog about design and user experience as well as generally working toward creating a more collaborative environment.

Customer collaboration over contract negotiation

This may be the easiest value to fit in to a user-centered design mentality. Since our work tends to be so user-centric, our focus is naturally on the customer.

There has also been a lot more motion in recent years around using the customer as a design partner or collaborator and there are many techniques to facilitate this. Personas are certainly a step in that direction, using a representation of the end user based on user research to inform design and influence decisions. Personas can be leveraged quite effectively within an organization to get agreement about target users and share the information in a digestible format that makes them easy to communicate broadly and even easier for the product teams to remember. Once the personas have become part of the collective consciousness, they become part of the conversation and ultimately influence the design. Indi Young's (2008) *Mental Models* describes techniques for modeling and visualizing customer behaviors, so that the user behaviors, and not just their wants and needs, can be communicated to a broader team with a goal of designing to these activities. The information to create the visualizations can come from observing or interviewing customers and can result in a very solid understanding of the user actions most important for the software to address. Collaboration can even be taken a step further by engaging the customers directly in design creation, using codesign activities or testing methods such as Rapid Iterative Testing and Evaluation (RITE; Medlock et al., n.d.). As user-centered designers, we are always customer centric, but there are always new ways to bring the end user into the design process; and this value gives us encouragement to work with our customers and to create an understanding among the team members that there is value in doing so.

Responding to change over following a plan

Many things can change during the course of a release cycle—technologies, market demands, user needs, and organizational structures. Things always come up that have the potential to cause the schedule to slip or feature work to be dropped. Even if there are no changes, work can often be more complicated or take longer to implement than originally planned. The longer the release cycle, the more potential there is for some kind of a disruption, especially within a product's competitive landscape. Sticking blindly to a schedule or roadmap in the face of evidence that the company should do otherwise is not a path to success. But it often happens, because a late cycle change in direction can be very expensive or result in an even later delivery of functionality. Of course, not all changes are dramatic enough to force or necessitate a major reshuffling of effort and resources. In fact, the most common adjustments teams are asked to accommodate are much smaller but can derail a release in a "death by a thousand paper cuts" fashion—the dreaded and, it seems, ubiquitous changes in scope or requirements. The need to make a change can occur for any number of reasons—time estimates were off and features need to drop to make the release date, new information came in from a key customer

requiring additional features to be added, or maybe a product manager simply needed to shift the priorities. Since these events occur more often than not, any good process should anticipate their occurrence, and Agile does that. This makes the process more adaptive and generally more able to achieve a successful outcome.

For the UX team, building in this kind of responsiveness to change can present an opportunity to incorporate more user feedback into the product. Ideally, some user research would have been done in the planning stages to help define the requirements, so that the starting point for the design and development work is on target. However, validation and refinement of the design necessarily occur during the development cycle, just as it does in more traditional development environments. A natural outcome of these activities is a list of changes to the design; although if the requirements and the user stories are well informed, the required alterations should be relatively small in scope. The difference when this value is in play is that the development team is more willing and able to respond to revisions in the design, because it is more ready and able to respond to change. It also helps that the team expects the possibility that it might need to respond to a new direction, not simply because history has taught this but because the process indicates that it will happen. In an Agile method, the frustration around dealing with change is mitigated, because activities are in place to support the introduction of new tasks or user stories that come from usability testing and events that allow these items to be given priority and potentially incorporated in to the release. It is easier to respond to change when the process provides a way to do so without disrupting the release. The process is also geared toward discovering these things as early as possible, so that their overall impact on the release schedule is much smaller.

AGILE PRINCIPLES + UX

"We follow these principles:
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need,
and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity—the art of maximizing the amount of work not done—is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. "

From "Principles behind the Agile Manifesto," AgileManifesto.org

Principle 1 - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software

This principle could not be more relevant, given that many products are constantly releasing updates and new versions of their sites or their apps to meet ever-changing customer needs and keep up with the constant improvements from competitors. For most teams, the days of 18-month or two-year release cycles are long gone or were never an option for their markets. Shorter cycles and increased responsiveness to customer needs do not always translate in to an increased focus on a great user experience. In the rush to delivery, features and functionality can still take priority over design. However, customers are much more demanding about the quality of the design and much less patient with features that are put together haphazardly. Expectations have changed to the point where users have little patience for software that is difficult to use, no matter how complex the application is. Not a designer among us has not been asked by an internal or external stakeholder to "Make it like a video game," or "It should be easy to use like TurboTax," or the newest favorite comparison, "Make it work like my iPhone." Whether or not these analogies are appropriate for a particular software project, and generally they are not, is less relevant than that the experiences people are having on a daily basis change the way we all think about software. So much good design is everywhere now and users have come to recognize it as important to their experience and expect a certain amount of usability from all their experiences. Along with this increased sophistication and awareness about ease of use comes the reality that the market responds to good design. This translates into an easier sell for the UX team to be a part of a process committed to generating

products that satisfy customer needs and have frequent releases to meet market demands.

Principle 2 - Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage

Changing requirements have been a reality as long as requirements have been written. Teams that recognize this and are encouraged to embrace these changes and view them as an opportunity are more likely to produce something that actually meets or exceeds customer needs. This Agile principle accepts that changes occur and acknowledges that they will come, as they often do, late in the release cycle. What is most exciting about this principle, however, is that it changes the mentality about this fact of life from resigned acceptance (or dread and fear) and positions it as positive and something that can be used to the benefit of the customer. This is so valuable for the UX team, that often findss itself in the position of delivering customer feedback late in the cycle and is often treated as the bearer of bad news. Instead of presenting change requests and knowing that only the smallest items, if even those, have a chance to make it in to the release or that everything will be deferred to an unspecified future release that never seems to come, with an Agile method in place, the changes are seen as the helpful information that it is. The change in attitude comes not just from accepting Agile values and principles but because Agile methods support this principle, by building in to the process a mechanism for dealing with late changes and ensuring that this information is solicited early and often. In traditional methods, feedback is deferred, because there is often no way to accommodate it without compromising the quality or the schedule of the release. Very often the entire team can see the value of the changes, but traditional methods created a momentum around the planned functionality that leaves the team a bit helpless to do anything differently. Not only does Agile support a different mindset with respect to course correction, Agile processes have mechanisms for accommodating these changes.

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

The frequent delivery of working software is of great benefit to the UX team. In more traditional environments, UX people generally have to work with rough prototypes or no working code at all, which may or may not be sufficient for expressing the complexity of an application. Even if a prototype was done by the UX team, it may not match the actual implementation unless someone on the UX team is directly responsible for building the front end of the UI (user interface). In a traditional environment, a functioning development prototype

is rarely available for review or testing until late in the release cycle. Generally, by the time working code is available, it is too late for usability testing results to influence shipping the release. Frequent deliveries of working code not only lends itself to more opportunities to do user research on the actual software, but shorter and more frequent shipping cycles mean that, when design changes are relegated to a future release, that day might actually come, since the next "release" might be only a few weeks away. When the software delivery happens more often, there is less pressure to squeeze every last item into the current release, since the customers will not have to wait years for the next version. This opens the door to allowing the focus of the product team to expand its focus beyond being exclusively centered on features and to consider the user experience as part of the release.

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

Of course, this principle would be even better if it referenced the designers, but if you read *developers* as "all functional areas supporting the production of the software," then it makes the statement even more powerful. Organizationally, the people who are connected to the business needs are often fairly far removed from the production team. Even if there is no distance between the business and production teams, there is often tension between the two areas. It is very rare for these two groups to work closely together, mainly because they have separate paths and few intersection points. Regardless of whether the issue is tension or distance, the dynamic contributes to a decreased awareness of and sensitivity to the business drivers on the part of the development team. Often, simply a lack of exposure and communication gaps create this lack of insight.

In one of my roles, I reported directly into product management and was amazed at what a difference it made to be a part of the conversations about business initiatives and product strategy on a daily basis. With that information in my consciousness, decisions based on strategic business goals seemed more obvious and product-specific trade-offs were easier to make. But, with their focus intentionally on the production of the software, development teams can find it hard to lift their heads up from the work of building the release to get involved in other interests, even though the purpose of producing the software is to meet the business needs. Often, it is difficult for them to access this information, even if they wanted to.

Usually, only the development managers attend high-level meetings where information is shared about business priorities. Contrast that with this principle, which has a simplicity and a genius to it, as it encourages a partnership with daily involvement between the production and business teams. It reduces the amount of communication necessary to keep each other in the loop by increasing the

frequency, it builds relationships, and it increases knowledge about the business for both sides. The two teams truly are partners, and not only recognizing this but fostering it is the healthiest thing for the product and, ultimately, the customers. The UX team benefits from this as much as the rest of the teams, since having an awareness of the business needs makes it easier to be more strategic about where we put our resources and attention. Left to our own devices, designers want to redesign everything that needs the help, and the list of potential candidates for design attention is usually infinitely long. It can be hard to admit that not every fix is worth making, but the more in touch the UX team is with the business needs, the easier it is to be aware of those needs. A product might contain some embarrassingly bad screens, but if only 2 percent of the users see those and they are able to perform their tasks despite the flaws on the screens, it probably is not worth spending design cycles fixing them. Just as the production team needs to be conscious of the user personas, it also needs to be aware of the revenue numbers and product strategy related to those personas.

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

One thing that is common, especially within the management teams of large development organizations, is to treat all staff as interchangeable resources instead of human beings with different strengths and skill sets. When looking at spreadsheet after spreadsheet of projects, level of effort estimates, and full-time-equivalent numbers, it is tempting to pretend that all resources are equal. It is often the case that availability is the most relevant factor in getting assigned to a project rather than interest or ability. This means that a developer who is much more comfortable and capable of tackling complex infrastructure issues may end up being responsible for creating a user interface for which he has no interest and possibly limited ability to write code. Finding the work to fit the skillsets of the team rather than trying to fit the team into the project at hand can be more difficult but can yield significantly better results and produce higher morale. Putting team members where they are best able and willing to contribute guarantees that more positive energy and enthusiasm is brought to bear on a project. This can be a boon for the UX team, as it is more likely to work with team members who are genuinely interested in and capable of producing a great UX.

The focus on creating an environment and providing the team with the support to accomplish its work clearly benefits everyone. This principle takes this concept a step further by positing the idea that trusting people to get the work done not only eliminates the micromanagement inherent in most traditional environments but also empowers the team to make the necessary decisions that affect the final output and make it better. After all, because a specification has the four to six required electronic signatures on it does not necessarily make it

any better or more relevant than something that the team sketched out on a whiteboard. The fact that design document was sent up the reporting chain, often to managers or executives who are quite distant from the project does not actually make that design any more effective at meeting user needs than trusting the team to make the right decisions. This principle assumes that, as part of their enthusiasm for the project and knowing that they will be allowed to take on the responsibility of decision making, the team has ramped up and done all the relevant user research, business requirements, market analysis, and technical constraints that affect the product. The team knows its project the best and is the most qualified to understand what should be done to make it successful. When team members are trusted to do that and feel that there is faith in their ability to execute on the work, the team will be motivated to work harder and prove this principle to be true.

Principle 6 - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

In an age of instant message tools, video chats, wiki pages, emails, and remote employees, there is often no need to interact in person with your coworkers anymore. Many companies take advantage of these technologies and a more global economy to have employees and team members scattered across the globe. However, nothing is more challenging to express over the phone than a design concept that is still in the early stages, and it is nearly impossible to collaboratively whiteboard a design via conference call. Not only will my colleagues miss out on seeing my excessive hand gestures, which admittedly convey enthusiasm more than any information about the design intent, they will be unable to grab a pencil or marker and join me at the whiteboard to hash out a concept and come up with a shared understanding of the design problem or solution. And rarely will an email thread, even one with embedded sketches, result in a common understanding and agreement to a design. Of course, more and more teams are distributed and travel budgets are growing smaller, so efforts need to be made to use technology to replicate the types of meetings from which a colocated team can participate and benefit. Certainly, if colleagues are remote, then digital sketching tools like Balsamiq combined with a screen sharing application like WebEx or GoToMeeting can absolutely replicate a sketching session in a conference room. However, if your colleagues are within walking distance of your cube, the more face-to-face contact you have with them, the better. Not only does this increase the efficacy of information sharing, it increases the sense of a team. It can be easy to ignore the blinking instant message icon in the corner, if you are busy. It is a little harder to completely ignore the blinking human standing in your cube. Also, much more information can be gained by an in-person discussion. Facial expression and body language can color an

interaction and transmit so much information that would be lost otherwise. Remote situations can be managed and work quite well, with a little creativity, but if you are not having frequent in-person conversations with the team members near you and opt to rely on technology, then you are building walls instead of relationships.

Principle 7 - Working software is the primary measure of progress

The siren call of metrics can be hard to resist. It can be so comforting to have all these numbers and graphs, providing you with tons of information about all the different things that need to be tracked as part of the release. The numbers can be followed, evaluated, plotted, managed, and discussed. Despite the best intentions behind most metrics, however, once any measure is put in place, people work to the number and not necessarily the spirit of what it is trying to achieve. But, at the end of the day, only one metric really matters and it is the progress of the production team—the existence of working software. Hitting the release date is also important, but the ability and likelihood of hitting that target becomes much easier to track when working code is available to asses and evaluate. The same is true for quality metrics, which are also very important but are much easier to accurately gauge when working software can be tested. In the end, you know you made the release date only when you have working software available on a given date, until then it remains at a level of uncertainty, no matter what metrics are in place.

For the UX team, this means that the measure progress can be evaluated by the incorporation of the research finding or design work into that working software. Someone once said that his UX team would be judged only by the designs produced and the designs that were implemented. At the time, I thought working on a team with this philosophy would be really freeing, because it would mean not being penalized for working with a difficult team that did not care about design and would be rewarded for having created wonderful designs even if they never made it into the product. Then, it dawned on me that such a philosophy also meant that a designer would not be penalized for being a difficult designer to work with or for coming up with designs that might be beautiful or represent a fantastic user experience but were impossible to implement. While I am very supportive of the idea of creating aspirational designs, when time and resources are tight, most efforts need to be spent on creating work that might actually make it into the hands of the customer. If working software is the primary measure of progress for the team, it makes sense that, for the UX team, the design and usability of that working software be the primary measure of their progress. This not only incentivizes the UX team to create a good experience for the customer given the technology available, but emphasizes the responsibility in working with team members to bring that design to life.

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

For some, it may be a surprise to see that the Agile principles include the idea of working at a sustainable pace, since Agile development cycles are have a reputation for feeling like a marathon run at a sprint pace (no pun intended). However, one of the original motivations behind the Agile Manifesto and its principles was to create a more hospitable work environment that did not routinely include 80-hour workweeks. The reality is that 80-hour workweeks are not sustainable and if that is what it takes to release a product, something is going wrong somewhere. Either the project is understaffed, has taken on too much work, or is grossly mismanaged; the bottom line for this type of project is that some issue that needs to be resolved. Ultimately, such an extreme pace leads to burn out and low morale, and neither condition is fertile ground for producing good software. While the rhythm of an Agile cycle is different than that of traditional methods and may take getting used to, if the pace is feels so intense that it seems unmanageable, then it is a sign that something in the process is broken and needs to be reworked.

It may be especially challenging for the UX teams get used to properly estimating the effort for small increments of work, but that is a skill that develops over time and with more experience in Agile. Until this happens, the UX team needs to engage in frequent communication with the rest of the project team and to discuss the issue in retrospective until the kinks are worked out. The goal is for the skill of estimating effort to be developed, and the UX team needs to understand how to load balance and size their work to fit into the new schedule. No one on the team, including the UX members, is supposed to work at a pace that he or she cannot sustain. This is not because Agile is a warm and fuzzy process; it is really a practical issue. If you are working around the clock to keep up, it means that you are overcommitting and you and the team do not have a good sense of what can be done in a given period of time. It is an indication that there is a risk for failing to meet deliverable dates looming in the team's future. While you may be able to push through and get the work done, the quality of the work may suffer as a result, or at some point you will burn out and be unable to continue taking on that much work. There are times when the workload spikes, for whatever reason, and if the team has been stretched beyond its means all along, it may not be able to accommodate an unexpected surge. It is better for the team to know what is reasonable and plan accordingly than to realize later that the expectations were completely unrealistic and were known to be so, but the team was not given the opportunity to adjust.

Principle 9 - Continuous attention to technical excellence and good design enhances agility

If we think of *agility* as meaning the production of high-quality code that meets or exceeds the customer's needs, then it is clear that using the best technology in the most elegant way and creating a great user experience design moves a team further down the path toward agility. What makes this statement so Agile, is the use of the word *continuou*s. Not only does it mean not just settling on a platform, technology, or a design then never revisiting the topic, but it means continually and constantly considering and refining these decisions throughout the cycle. This does not mean that a heavyweight desktop app should change course halfway through a release cycle and try to ship as an iPad application instead, just because it is a cooler technology. (However, it is worth mentioning that this kind of change in direction is not outside the realm of possibility. Such a major adjustment would only be likely to be done if a key customer or entire group of users switched platforms and the product would fail if it did not release on the right platform.) It does mean that opportunities to evolve the underlying technology or improve the architecture should be at the forefront of the development team's consciousness. It does not require that the team consider only what is being done for the current release but have an eye toward the future and consider whether the current version will support future possibilities. Similarly, the UX team should be looking around to see if new technologies or design paradigms might allow it to improve the design for the customer. And, as the team moves through the release, the UX staff should be looking for opportunities to improve the design. These improvements can be inspired by customer feedback, conversations with developers and internal stakeholders, or witnessing an innovative and relevant implementation. Unlike more traditional methods, the design is not done when the specification is completed and put on a shelf, with no real way to revisit it until the next release. The UX team needs to look at the design as a living thing that can and should be revised when information is available to improve it and provide a better experience for the customer.

Having a principle that speaks to the importance of continuous attention to good design makes the job of a UX person that much easier. After all, if I had a magic wand to wave at work, this might be one of the things I would wish for. It might be true that the original intention of the use of the word *design* could have been to speak to the architecture of the code, but we can certainly broaden the context of the statement to encompass all the work that goes on to produce the front end. As with the focus on technical excellence, it is the element of "continuous" that makes this principle so interesting and so powerful. In more traditional methods, design can often be talked about at the kickoff of the project, at a certain milestone, like the delivery of the

specification, or as a last minute addition once the code is written. This is less likely to be true with using the Agile method, since the work is constant and ongoing. However, saying that the attention to good design should occur throughout the process recognizes the need for it to be a deeply embedded part of the work and not something smacked on when the team thinks about it. To achieve a truly great design, the effort needs to be an integral part of everything that the team does.

Principle 10 - Simplicity—the art of maximizing the amount of work not done—is essential

Simplicity is the most valuable concept in design, and it seems appropriate to apply it to the work process. For the UX designers, this might not represent a great challenge in terms of changing our mindset, because we so often shrink to fit into the time we have available and often do it by focusing on doing only what is most necessary. The best thing about this principle is that it is a call to action in terms of being very conscious about where those on the UX team spend time, what they produce, and if there are places where we could spend less time or do things differently and achieve the same or a better outcome. This does not mean just doing less; it really means spending time and resources where they are the most effective and being as strategic as possible in choosing the activities in which we engage. By all means, if having that second brainstorming session did not yield much, then the next time you should consider whether a single brainstorming meeting might be sufficient. If you and your team to discuss what is and is not working and actively examine how you spend your time, the extraneous tasks are often easy to identify. Generally, your teammates will not hesitate to let you know when you have had one too many brainstorming sessions. Some areas of waste are not quite as obvious, because the only time being wasted is yours, and your teammates might not be as quick to flag it as a problem. It is important to be very conscious of the impact your work has when looking for opportunities for efficiency. If you spent time coming up with a design solution with a developer and he is already well on his way to implementing it as agreed and you then spend your time producing sketches or wireframes to reflect that agreement, you may want to ask yourself, Why? Is it out of habit or the feeling that, if you did not write it down, then it did not really get designed? If so, then it may not be time well spent and you should consider saving your effort. If the answer is that you did it so you could communicate the design to a QE (Quality Engineering) team in another country, then it might be the right thing to do, but make sure that you are producing only what you need to in order to provide that team with an understanding of the UI. It is important to look at everything you do and ensure that you are being as simple as possible in your efforts.

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

Much like the idea of building teams around motivated individuals and trusting them to do the work, this principle is geared toward allowing the teams the freedom to make the necessary decisions to achieve great software. The team should be allowed to control the timing and flow of its work because, as the people closest to the project, the members best understand the right course of action. Additionally, the team is in the best position to know which person is the best choice to execute a given task. For the UX practitioner, this can also mean being more open to working collaboratively with other team members on design and allowing the design work to be supported by those whose are able to execute it. This certainly does not mean treating design tasks in a random or haphazard way but recognizing that, if a developer has a good eye for design or creative implementation solution, then it might make sense for that person to handle those decisions for a particular piece of the project. Perhaps, you can seed the effort with some discussion, then let the person take it from there. The UX person always is responsible for the outcome, so this is less about dumping your design work on a willing victim and more about allowing people with the interest and skill to contribute to the design in a more meaningful way. This can give the team a greater sense of ownership with respect to the design, and it can help maximize what is most likely a limited design resource.

Principle 12 - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

It may be the last principle, but I see it as the most critical. Without this, there can be no successful Agile implementation. These moments of reflection, done as a group or individually, allow the team to work more efficiently. This evaluation is especially important for teams beginning their Agile journey, as it is necessary for them to identify and learn as a group what is or is not working, the need for this does not go away when teams get more experienced. The only difference is that the necessary course corrections or process tweaks might be smaller. It is not realistic to think that a team can be completely efficient and Agile as soon as the project kicks off, with no need to examine how the members work as their way through the release cycle.

In looking at effective teams, it seems that the ability to do this well is the best predictor of a positive outcome. The first retrospective session that a team has may certainly be awkward, but if the team is engaged and the conversation generally heads in the right direction, the odds are pretty good that it will not take long for the individuals to master this skill and ultimately find success with an Agile process. However, if in the first retrospective, team members'

conversations are shut down or they feel like their feedback is not being heard, it can be tough to recover from that first experience. The team will have a much harder time refining its effort, because the members have no a forum in which they can express their concerns and work through solutions. If that dynamic continues in the retrospectives, not only will the team miss out on their best chance to improve its process but this could be a red flag that other communication issues are going on and need attention. The scrum master is responsible for making sure that this meeting is productive, but depending on his or her background, this person may not have to skills to get the discussion moving in a healthy direction if it is not doing so naturally. As experienced facilitators and moderators, it is the responsibility of the UX team members to help make the exchange as fruitful as possible. In an official scrum retrospective, the scrum master runs the meeting, but that is not to say that the UX person cannot help guide the conversation in a positive direction. If a discussion is about how the process is going is being done in a more informal way, then it may be easier for the UX person to jump in and manage the tone of the discussion.

COMMON METHODS

Crystal

Crystal refers to a family of methodologies developed by Alistair Cockburn that are color-coded based on the size of the team. It is one of the first entrants into the pool of Agile methods, since it was created even before Extreme Programming and given a formal name prior to the creation of the Agile Manifesto. In terms of method and process, this is probably the most lightweight and flexible of them all. Rather than create a series of ceremonies or events, it focuses on addressing risk on teams and using a process that allows the development team to achieve certain characteristics by using a set of methods, some of which are identified as optional. It is a framework that offers some specific tactics and techniques and is more specific than the principles of the Agile Manifesto but less prescriptive than most other Agile methods. The best source for more information about how to apply this framework to small teams would be *Crystal Clear* by Alistair Cockburn (2005).

Extreme Programming (XP)

Extreme Programming is one of the earlier Agile methods, and it remains very much a part of the current discussion of Agile. It is worth mentioning not only for its historical significance but also because, for many developers, this method is often their earliest and defining experience with Agile. They may have moved on it using other Agile methods, but this is the technique that sets the tone for their understanding of how Agile processes work. One of the most interesting elements of this method is the "Customer Bill of Rights" and the

"Developer Bill of Rights," which introduced the concept that the process of producing software needs to take in to consideration its obligations to both its customers and the people who are writing the code. It is a fairly detailed development process, the main purpose of which is to empower the team to take real ownership of the project, encourage frequent communication among team members, participate in honest planning, incorporate customer engagement into the process, and acknowledge that iteration is a desired part of the process. The emphasis on refactoring seen in all Agile methods was born from this practice.

This is a very well-defined methodology with a very heavy focus on code production to the exclusion of considering the roles of other functional areas. This is not a flaw in the method; it simply is meant to address the faster, higher-quality production of code and speaks to that part of the production process. A UX team should be prepared to be highly communicative and integrated with a development team that will move rapidly to find solutions and implement them. There is a high likelihood that there will be access to working prototypes to use for testing very early on, when working with this method. Additionally, if the developers have really embraced the values of this approach, there will be an openness to rework and refine the design, as refactoring is highly valued in this technique.

Scrum

This is one of the more popular of the Agile methods right now. The two key features in this process are the roles of the team members (the product owner, the development team, and the scrum master) and the product backlog. Each project team has a scrum master, who is responsible for moving the process forward and removing obstacles that might prevent it from doing so, and a product owner, who is responsible for setting the priorities of the work items. The team itself also plays an important role, as the members' consensus on and commitment to a given scope of work is an integral part of the process. The team commits to contents of a sprint, after discussing each element and estimating its required effort. Should a need arise to change that scope or timing, only the team can agree to make that adjustment during the course of the sprint. Of course, the request to change scope or direction in mid-sprint can come from within or external to the team, but since the team has made a commitment to do a specific amount of work within a specific amount of time, only the team members can decide to alter that commitment. In practice, the issue around changing direction during the course of the sprint does not often come up, but there is value in knowing that the team controls its own destiny, at least for a set period of time. In an environments where the scope of the development work can change on the whimsy of executives, creating a safe place, where the work cannot be disrupted, allows teams to focus in a way that they might not have been able to do in more

traditional environments. The product backlog is the repository for the user stories, the tasks needed to satisfy them, bugs, and nonfunctional requirements. Anyone on the team can contribute to the backlog, although a larger team needs to determine the priority and timing of the work. The product owner represents the voice of the customer, owns the priorities of the work items, and works with the team to define user stories.

Scrum also has many events geared to facilitating productive and frequent communication between team members. There is a daily scrum meeting, also called *daily standup*, which is a recurring 15-minute meeting. While anyone can attend, only the contributors can speak, and they are meant to share only what they did yesterday, what they intend to do today, and what obstacles they face that might inhibit their work. Some organizations may also have a daily scrum of scrum meeting, where representatives from multiple scrum teams meet to discuss work that overlaps or affects the different teams. Both events provide predictable, daily opportunities for team members to engage in conversation and facilitate communication in an efficient way. Also, events occur for every sprint cycle as part of the maintenance and planning for each sprint. There is generally a backlog grooming session, in which the team estimates the effort for a given story, refines and negotiates the acceptance criteria for a story, and breaks down large stories, or epics, in to smaller chunks. This meeting is supposed to be only an hour long, but more than one of these may be necessary for a given sprint, especially at the beginning of the release cycle, when more decisions are made about what work needs to happen and when it should occur. Sprint planning should also take place at the beginning of each sprint, the first day of the sprint, so that the team can scope out the work and commit to a specific set of deliverables for that sprint. This is the longest event in scrum, time boxed at a maximum of eight hours. Smaller or more focused teams may find that they do not need that much time for planning; larger project teams may have to work harder to stay within that time frame. At the end of each sprint, there should also be a sprint review and a sprint retrospective. The sprint review is where the team discusses what work was or was not completed, according to its ability to meet the acceptance criteria and definition of *done*, and demonstrate the results of the sprint to the team and stakeholders. The sprint retrospective is where the team acknowledges what went well in that sprint and discusses what did not go well and how to do better going forward. Fostering a candid and open forum in this meeting is critical to making scrum a process that works well for a project and for the team. This is the event that requires the most open, honest communication; and if that type of discourse is not facilitated or the feedback is not acted upon, it can inhibit the ability of the team to work well together.

This approach is not very prescriptive where design and usability testing are concerned, so it provides a great opportunity for the UX team to be proactive

about defining its role in the process. Depending on the scope of the project, the dynamics of the team, the length of the sprints, and when the designer gets on board, you can decide if it makes the most sense to work within the sprint to produce the deliverables. It is not uncommon for designers to work ahead of the development teams to define the interfaces and gather feedback while still maintaining constant communication with the rest of the team. Working in this fashion is also referred to as *working in parallel* to the development team. When the design team works independently of the development team by working on a design problem in a separate sprint from its implementation, this can give the design and research teams a little more room to breathe. They can determine their own velocity and have the entire length of the sprint or sometimes multiple sprints to work on their design solution or to complete their testing and analyze the results before the implementation begins. If the UX person is working within a sprint, along with the rest of the team, he or she generally has less time to work on a given item before it needs to be coded. This can be fine if the designer is partnered very closely with a developer or is working on a very small piece of work. For user research or when addressing larger design issues, to do so in a different sprint than the implementation tasks can often be advantageous. While some UX teams work exclusively in parallel to or within a sprint, it is certainly possible to mix both techniques over the course of a release. When thinking about how to include user research in the cycle, it helps to decide at the beginning of the release if doing weekly user testing is possible or if dedicating a sprint or two to usability sessions is more realistic. Larger pieces of research ideally are done ahead of the cycle, if possible, or in an early sprint or sprints, if there is no time to research ahead. What is really critical is to thoughtfully consider what the UX role should be in the framework of a specific team's implementation of Scrum and with the team dynamics in mind. If it is your first time participating in Scrum, just take your best guess at what has the best chance of success based on the culture, the team dynamics, and the project and be prepared to iterate on that approach.

Hybrid agile, or custom agile

Hybrid Agile, or *Custom Agile* is used to describe what happens when a company takes the elements of any Agile methodologies that are appealing to it and uses them in a way that makes sense for the organization. The company may do this because only a part of an organization is using Agile methods and the efforts need to conform to a broader project management framework. Or, perhaps, combining Agile methods with traditional methods is just a good fit for the company and the culture. This requires a project team to be conscious about how its adoption of Agile methods fits the broader framework and what striking this balance means for the team. Certainly, more effort is required in doing this, because a team cannot use a prepackaged process and will (ideally) invest time

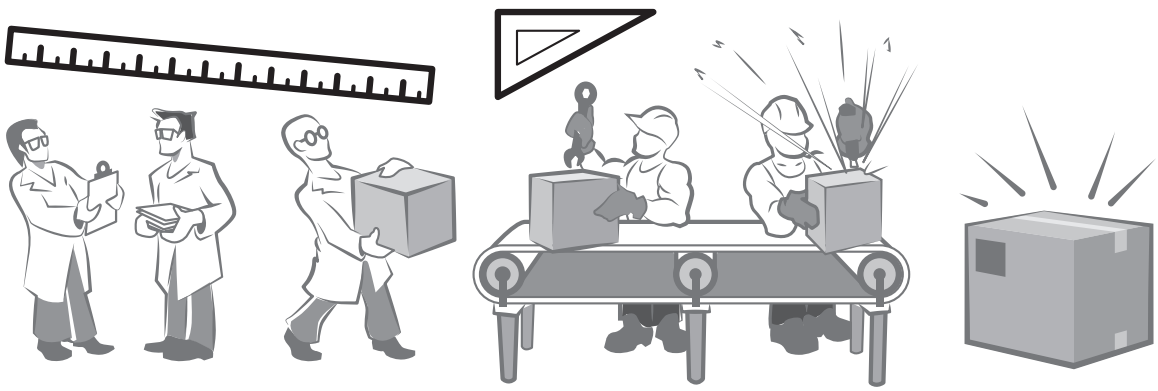

FIGURE 1.2
Custom agile.

in making the best trade-offs. If this investment is made, then creating a custom process can potentially increase the odds of a successful and less painful transition in adopting Agile. There is a potential for trouble, however, if an organization is not being conscientious in defining this process—it can wind up with something that is called *Agile* but is more or less the same traditional process always used. This outcome is especially likely if an organization is hybridizing its Agile approach because it is simply reluctant to make a broad organizational change. This type of approach tends to have a high rate of failure, because it can often reflect a lack of genuine commitment to Agile rather than fully embracing the concept, instead just trying to squeeze it into the existing culture.

If the organization is genuinely committed to customization, then this type of implementation can be the most open to allowing the UX team to define its own role and be a very active voice in the process. If there is dedication to defining a homegrown method, the process definition should include all the functional areas involved and account for their needs in a way that fits best for the organization If your organization has taken this tack for adopting Agile, the odds are pretty good that stakeholders from all areas will be involved and the result will be something that works well for all the teams. The good news is that a culture that embraces this approach will likely get it right or be open to course correction if it did not. On the other hand, if an organization is choosing this approach because it is hesitant to give up processes that are old, comfortable, and well established or there is a sporadic application of Agile throughout the organization, this method can be the most challenging approach of all. In is unlikely that much training will be available to people when that kind of dynamic is at play. The company might train a few key people but not see the value in providing training for something it feels will be defined in-house. This

can be problematic, because it is very hard to know where to make changes if you lack a good understanding of that from which you are deviating.

Another potential issue is that this might mean that a UX person can find himself or herself supporting both Agile and non-Agile projects, which can be a scheduling and logistical nightmare. The design team can also find itself working with all of the urgency and speed of Agile but still producing heavy-handed specifications and experiencing the worst of both worlds. Another problem that often arises under these circumstances is that each team has a different understanding of Agile; and if the UX person is supporting multiple projects, it can be a challenge to know how a given team is working. It is hard to generalize knowledge of an hybrid method if no single method is in use.

Kanban

The Kanban technique is not as widely adopted as Scrum but has definitely gained traction recently and seems to be part of the next wave of Agile methods. The term is derived from the Japanese for "visual card" and is a relative of Toyota's Lean production processes. The central element of this method is a formalized version of the type of board often used by Agile teams. On a Kanban board, there is a "Goals" column on the far left that describes the team's goals, to provide a reminder of the overall purpose of the project. A "Stories" column contains all of the stories that will be worked on, and a story remains in this column until it moves through the process toward completion. Each subsequent column represents the work steps necessary to get to the final state. These columns are defined by the team and could include "UI design," "Development," "Validation." The top part of each column is for stories in progress for that step, the bottom is for stories that are complete for that step but have not yet moved to the next step. The final column holds the stories that are complete and of shipping quality. For this method, a little less structure is given to the time it takes to complete a task; the task moves from one column to the next as it is ready. This can allow for a more a natural cadence for the team, as the work can flow as it needs to, in contrast with Scrum's sprint cycles and formal commitments.

It is very important that UX activities are included in the column to create visibility around the importance of those tasks and the necessity of completing them for a task to be "done" or move forward. Design work always warrants having a column; and it can be very powerful to have the visual reminder that design work should precede implementation. It can also be educational for the team to see how long a given card spends in the design column. For that reason alone, there is value in having design as its own column. If other UX activities, such as usability testing, are only going to happen for a few features or will be done infrequently, then they might not require their own column on the board. Another challenge for teams working in a Kanban environment is that often

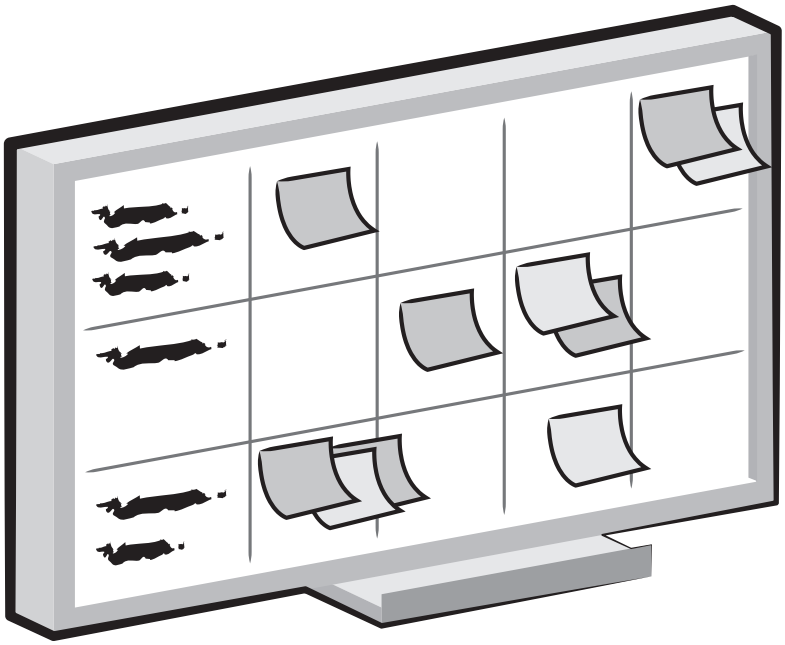

FIGURE 1.3
Kanban.

seeing the "big picture" across projects is difficult, since the Kanban boards are for a single project and very local. Of course, when this happens, it is not just a problem for the designers, but it can make their lives especially challenging if they support multiple project teams and do not feel that they have an adequate sense of how all of those projects fit together.

Since this process is, in fact, very light on process, it works best in environments where the team and the culture already support a high level of communication and collaboration. One of the reasons that methods like Scrum exist is to provide a framework that facilitates the creation of a highly communicative culture and to provide events that create the opportunities for the team to realize Agile values. Choosing to use Kanban should be done when your team does not need these cultural training wheels and is ready and able to simply execute the work using a lightweight process to give the work some structure.

Scrumban

As its name might imply, this is a hybrid of Scrum and Kanban, the term coined by Corey Ladas (2008) to describe the method that his team developed. It is essentially a leaner form of Scrum but does away with some of the events typically associated with Scrum and uses the Kanban board to visualize the

project; however, it adds a value to the cards to show their relative weight within the project. To learn more, read Corey's paper online at http://leansoftwareengineering.com/ksse/scrum-ban/. The idea of adding weight seems interesting, as it can help to expose and highlight some of the complexity that might not be as obvious on a Kanban board, where all the tasks look the same. At the same time, it can also provide a little more structure for a team that is either transitioning from Scrum to Kanban or for those teams where this type of approach simply fits their needs.

Lean UX

This is another of the next generation Agile methods and the first one to consider UX explicitly. While not exactly in its infancy, Lean UX is still very much in its youth and is gathering an increasing amount of momentum and attention. The term was born as a result of Eric Ries's (2011) *The Lean Startup* book and movement, which encourages rapid prototyping and user validation and the continuous deployment of production code in response to a fast-moving market. The idea of integrating design into this process led to the creation of the idea of Lean UX. While there are several definitions and many strong opinions about what Lean UX is or is not (and more than a few books being written on the topic), all of the current definitions seem to have a few commonalities. As its name implies, there is no waste in this process. For some adopters, this means no deliverables at all, for others it simply means no unnecessary deliverables and few products from the UX team. This process moves fast, so it needs to be very streamlined, and the focus of the UX team needs to be on influencing the design and processing customer feedback to guide the next round (or current cycle, if possible) of designs. In some ways, the emphasis in the name on UX is almost misleading, because a Lean UX team must be seamlessly integrated into the production team to be as efficient and effective as possible. While UX is a prominent consideration, it is most certainly the method that is the least likely to have any silos.

The other element that everyone agrees on as being a part of Lean UX is frequent and continuous feedback from customers. These types of UX teams are the most likely to have found a way to incorporate weekly customer testing sessions into their cycles, despite also generally having the fastest-moving and shortest production cycles. They have recurring usability testing sessions, which often occur in person. In addition, these sessions are often complemented by remote or automated web testing tools that gather feedback. Teams that lack access to local users rely on the remote techniques, as these still yield valuable results, even if they do not offer the same kind of face-to-face contact. The team plans the recurring sessions with customers, and the topics can range from getting feedback on concepts or wireframes to full-fledged user testing. The concept of "test what you have" is very common among Lean UX teams. While I

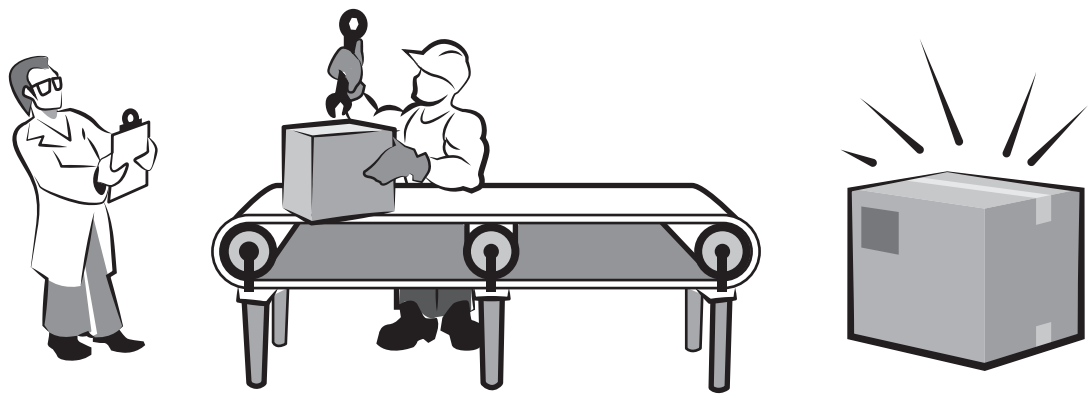

FIGURE 1.4
Lean UX.

have not heard anyone explicitly include this in their official Lean UX definitions, I notice that these teams are also the most likely to have very organic, highly adaptive ways of working with the other functional areas. This makes sense, really, since these teams need to function at such a high level of efficiency. It can almost be challenging to talk about "process" with Lean UX practitioners, because so many of them have managed to fall into a very natural and almost transparent rhythm of working, to the point where the artifice of process just falls away.

COMMON TERMS

Chickens and pigs

Every project has people in the role of chickens or pigs. The chickens are the team members involved in the project, but not necessarily directly, and need to be informed of its progress. The pigs are the team members committed to doing the work and are directly responsible for the outcome of the project. The terminology comes from a description of a bacon and egg breakfast, in which chickens are involved, but pigs are committed. The UX team usually falls in to the pig category since it directly contributes to the outcome of the release and is "committed."

Product owner

The product owner is the owner of the product backlog and the deciding authority on priorities and user story definitions. Which functional area fills this role is dependent on the organization, but typically it is a product manager or other type of business owner. It is not unusual for

a development manager to be in this role, if the organization has no product managers, the product managers are in another location, or the product managers are not usually involved in the release in a hands-on way. It is uncommon, but not unheard of, for someone from the UX team to be in this role. Regardless of the functional area from which they come, the UX staff members should expect to work as closely as possible with whomever does fill that role. One reason that it is critical to establish a good working relationship with the product owner is because that individual is ultimately responsible for the shipping product. The UX person is responsible for the creation of the design, but the product owner actually "owns' the product, which includes the design. Since the product owner oversees the assignment of priorities to the user stories and the work in general, it is important that he or she understand the value of user-centered design so that those efforts are not constantly put at the bottom of the priority list. In general, the product owner works well with UX, as that individual appreciates the research produced by the UX team and is happy to know that someone is looking after the design.

Scrum master

This is the person responsible for keeping the Scrum process on track. As the person responsible for facilitating the removal of impediments raised in the daily scrum meetings, he or she keeps the meetings on track and on time and generally is responsible for enforcing the agreed-upon Agile process. The non-Agile analogy would be a project manager, and the role tends to be most effective when it is filled by a neutral third party who is not directly involved in the production of the releasing deliverable. Scrum masters act as time cops for the scrum events that are time boxed; they facilitate the retrospectives and generally make sure that the team follows the process it has chosen to follow. Ideally, however, scrum masters do not act as the "scrum police," which is not an actual Scrum term but my own description for scrum masters so enamored of the Scrum process that they cannot see the forest for the trees. It is not their role to be rigid about following Scrum to the letter and in the way they saw presented during the scrum master training. At first, the scrum master might well insist the team does follow the process exactly, so that team members can properly identify where they need to make change. This initial adherence to the formal process is entirely appropriate and of benefit to the team. However, if the team agrees that something about the process does not suit it and identifies an alternative approach, the scrum master should be open to supporting the team in that change and keeping them on track with it. He or she should not declare the change to be "not Agile" and try to talk the team out of doing it. This role is really that of a facilitator who has a good working understanding of Agile processes and helps the team get the most that it can out of working in an Agile way.

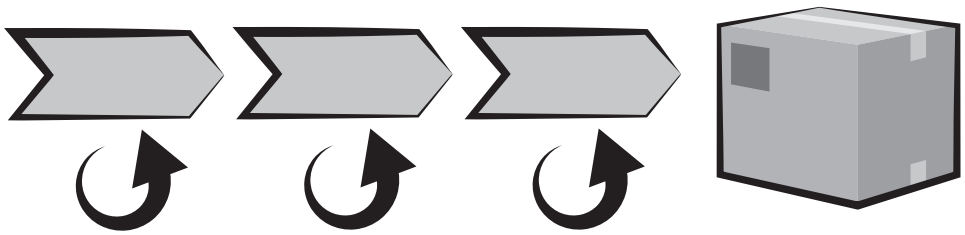

FIGURE 1.5
Sprints.

Sprint

Essentially, a sprint is single development iteration in a given release. Each sprint has a scope to which the team commits and its goal is to have demonstrable software at the end. A single release cycle is made up of a series of sprints. A sprint's length is typically between one and four weeks, with a three weeks being a nice balance between meeting time and work time. Most teams use a consistent sprint length throughout the release, unless some need arises for adjustment as the team moves through the development process. Teams using Kanban or a less formal Agile method may not use a specific sprint length and allow the work to follow its own pace. The estimation of the level of effort, using the abstract of story points, are tracked to determine the team's overall velocity. The velocity helps determine how much work the team tends to complete in a given sprint, so that it can plan the appropriate scope to take on in future releases. The UX members can be tracked as part of the team's overall velocity, which may or may not be helpful to the process. Each team should determine the value of this work and if it is helpful to the greater team. It may be interesting to the UX person that the team are 10 hours into a 20-hour design task, but if that person is straddling multiple projects that 20 hours could be evenly spread across the sprint or occur all at once at the beginning or the end. This might not help the team or the UX person know if the team is on track for the sprint, especially if all of his or her work occurs in the last few days. Regardless, the issue of tracking UX velocity should be discussed by the team and agreed upon ahead of time, although it can be revisited in a retrospective if necessary.

Product backlog

At a high level, the product backlog is a prioritized list of things related to a product, typically for the current release. Because the team reviews the product backlog often, it may be a distraction for the backlog to contain items that are out of scope for the release at hand. Some teams manage this by coding these items with an exceptionally low priority and a specific numerical value, so

that they can remain available for consideration in the event of a schedule slip or other dramatic change in circumstances. If the items never make it in to the release, the team may roll them into the backlog for the next release. Other teams create release-specific backlogs and move out-of-scope items into the backlog for the next release or review in the future.

Items in the backlog can include epics, user stories, tasks, or bugs. When the team does its planning, it pulls from the backlog to inform the scope of the release. When the team engages in backlog grooming, it reprioritizes items, fleshing out acceptance criteria and refining the estimates. While the product owner is the ultimate owner of the backlog, it can be very helpful if the UX person is part of the process of creating the backlog. If any high-level design has been done, the UX person can make sure that this perspective is supported by the user stories and tasks in the backlog.

User stories

Instead of functional requirements, which can inspire a myopic focus on features and functionality, Agile methods rely on user stories, in an effort to keep the attention and awareness on the needs of the user. User stories have a consistent format that phrases everything in terms of the users and their needs—"As a …, I want to … ." Additionally, "in order to…" can be added to the end of user story to provide more insight into the motivation behind the story, but the shorter format is the more typical of the two. While its helpful for the team members to be involved in the creation of the user stories, UX people tend to be more accustomed to thinking of the user perspective when considering functionality. Strong UX participation during the population of the product backlog is also the earliest opportunity to influence the process and incorporate user-centered values into the work well ahead of executing any design tasks.

A good example of a user story is "As a mom of an active preschool girl, I want to see all the skorts available in a size 5T so that I can buy as many as possible." This describes who the customer is—a person shopping for someone else. What that person is doing—looking at a list of available skorts (skirts with built-in shorts) in a specific size. And, if your team takes the extra step, the story ,explain why the user is doing the task—to purchase multiple items. While not every team uses the "so that I …" part of the sentence, it is clear that having this extra phrase can be very helpful, especially for designers. A person who wants the list to compare prices against another site, identify the fabric content, or get a sense of how much variety is available on the site is really performing a different task than a person who wants to identify all available items and buy multiples of them. It could mean the difference between having a list of thumbnails with prices and a way to select multiple items to go into the shopping bag

or having a broader list that also includes product details. If the various solutions for different motivations have commonalities, as they do in this case, it might make sense to provide a broad solution that will address several different "whys." Having the discussion with the product owner around this part of the user story could be very valuable, especially for the UX team. Knowing the "why" might lead the team to a different design solution than if it has to fill in the blank itself. Having it explicitly defined also ensures that everyone has the same motivation in mind, which can produce a more consistent outcome.

Epic

When a user story is so large that it cannot be completed in a single cycle or is defined in such a way that it needs to be broken down further, it is considered an epic. An epic needs to be broken down into more digestible pieces, which can potentially fit within a sprint cycle. It might seem like the best thing to do is to simply decompose the epic in to its smaller pieces and get rid of the high-level item once it has been identified as being too large to fit. However, preserving the larger story allows for relationships to be established between the smaller items and provides a more obvious picture of how things fit together. It also provides clarity about when this particular requirement is really met, if only four of the five smaller pieces have been implemented, then the epic has not been satisfied.

Planning poker

One technique that can be used to estimate effort for a user story is to give each team member a set of planning poker cards. In this way, team members can put up a card giving their estimate of effort, people who have a higher or lower estimate can explain their estimates, and the team can decide if an adjustment needs to be made. The outcome of the subsequent discussion is an estimate for relative size of the story. It is important for the team to be clear on what it is estimating, whether the overall team effort or just the effort for that functional area. Otherwise, a team can waste time doing a Delphi on the estimates, such as if a UX person is asked to estimate development effort or vice versa. Part of this clarity requires a decision around how the UX work will be handled in general. A user story should be applicable to all the functional areas, as it is really just a specifically formatted customer requirement. A task should be specific to the functional area, and if dependencies require a UX task related to a given story to be done prior to the implementation task for that story, these dependencies can be captured at the task level.

Story-point estimation

When estimating effort, most methods recommend using story points rather than hours or days to determine the relative effort of the work. It can be

a challenge for teams to think so abstractly about effort, and if numbers are used (rather than small, medium, etc.), the natural tendency is to try to map a story-point number to hours of work. It is best if the team can fight that urge and instead determine the baseline of the story points against a specific task with which everyone is familiar to establish that a task like *x* is small, a task like *y* is medium. Once a baseline is established, the team can use these as a reference when trying to gauge the amount of work required to complete a particular user story. It can be quite difficult for team members who are more comfortable with precision to become accustomed to estimating effort in such abstract terms. However, the point of using a concept such as story points is to keep team members from getting caught up in a false precision during estimation. While it is certainly easier to conceive of what 10, 20, or 40 hours of project work looks like, that does not make an estimated effort of 10 hours any more accurate than a story-point estimate of small. If you think about the last estimation effort in which you tooka part, a certain level of abstraction was probably applied to the numbers; and you can see that what was perceived as a small task would always be estimated at 10 hours.

Estimation of story points is something that needs to be considered when defining the role of the UX team in the process. If the UX team is going to be on separate sprints, it may or may not be important to estimate its effort and track its velocity. Since UX teams are always expanding or shrinking to fit the time they are given, putting too much effort into estimating the effort of UX tasks might not be a great use of the team's time. However, if the UX team uses parallel sprints for its work, it might be interesting o track velocity to help it with its own planning. In most cases, the UX team can do its estimates solo and let the product team review them if interested. This is especially true if a UX person supports multiple projects. In which case, that individual's velocity is determined more by the time is available to be spent on a given project rather than how long it takes to do a particular task or tasks. Additionally, when it comes to estimation and tracking the velocity of user research, there might be nothing interesting or insightful to track. Six hours of testing takes six hours; and while some scheduling or posttest analysis might occur, these are pretty discrete events and estimating them may offer no value to the team. These tasks have no significant amount of potential for variability that represents a risk that requires management. It is probably sufficient to account for the time and know that a day or two days of testing activities occur at a certain point in the sprint. However, if the UX team needs to track effort to load balance or manage staffing resources or the product team wants keep track of UX efforts for some other reason, it may be useful to estimate user stories in a way that accounts for the UX team's time. As with the other elements of the process, it is important to work with the team to define the approach to this up front and revisit it as necessary throughout.

Acceptance criteria

These criteria determine what needs to be included in the working code for a given user story for it to be considered ready to ship. It defines the scope of that user story in terms of functionality and behavior for both the user and the system. For the user story "As a doctor, I want to see an overview of my critical activities, so that I can take action on them," the acceptance criteria might include "the total number of critical actions is displayed," "an overview of critical actions is displayed," and "the overview links to each critical action so they can be resolved." This way, the product owner can communicate the details about the user story, the designer can add his or her perspective, and the developer can have a better understanding of what the user story intends to achieve. It also allows the team to agree on what the implementation will contain to be considered shippable, so it represents a tacit agreement as to scope. Defining this ahead of doing the work also avoids difficult discussions later in the cycle. Often, it is not until the code has been written and submitted that team members realize that it does not quite do what they had in mind. Even in a responsive Agile environment, waiting to have that conversation during the team demonstration or just prior to the end of the sprint puts everyone in an awkward position. Either the work is not considered complete, which causes the team to miss its deliverable, date or the rework is pushed to the next sprint. Doing the rework later is fine, but it could have been avoided with a little more discussion at the beginning and a more explicit expression of expectations.

Burn-down chart

This chart is used in daily meetings to visualize the overall progress of the team and is a plot of the effort against the remaining hours. Ideally, this is a nice straight line down, as the amount of work left to do decreases throughout the sprint. In reality, the actual burn-down chart often has many spikes and valleys, which ideally have an overall downward trend. To track effort, the tasks must have estimates associated with them, and the team must track the completion of and time spent on each task. These charts tend to be most valuable for the development team, whose work occurs consistently throughout the sprint and the release cycle. It tends to be less insightful when used for tracking QE or UX efforts, as those may happen ahead or behind certain tasks and take place predominantly at the beginning or end of the sprint. It may give the team a picture that indicates less progress than is actually true, if the burn-down chart routinely includes tasks that typically happen at the end of the sprint, such as some QE testing tasks. Similarly, if the UX team is working in a sprint and its design tasks take place in the first few days, the burn-down chart may look a little more positive than it necessarily is, as the designers mark their work as complete. If the charts are reviewed by only the project team, this is not much

of an issue, as it can take these things into consideration when reviewing the chart. However, if people outside the team are looking at these plots, they may not understand this and get the wrong message. When considering how to use these charts and generally approaching how to track velocity, the team should consider its audience.

Spike

Some teams use spikes to manage stories that cannot be estimated without some amount of investigation. Given the short time frames of sprints and the Scrum emphasis on committing to a specific amount of work, it makes no sense to simply sign up to do work of an indeterminate duration. Under any circumstances, tracking the progress of an effort that lacks an expected duration is hard, so a technique was created to address this problem. To deal with a spike, the team is given a set amount of time to do the necessary research, and the outcome of the spike is an estimate for the user story. Spikes can be used to accommodate UX design work or to account for user research, but the method is really intended to handle uncertainty around implementation solutions. Design and research should be predictable events that are a part of the normal planning. Spikes should be used to manage exceptions, such as a design task that requires research into available technology before it can be estimated or unexpected issues that come up and need user research to be properly defined.

AgileFall

While this may not be an official term yet, its use in conversation seems to be gaining popularity. The term refers to the situations in which Agile methods are fit into a waterfall organization or when a sort of mini-waterfall process starts to occur within an Agile environment. Sometimes, a team may be practicing Agile and the rest of the company is not or cannot, and the team strikes a balance between the Agile teamwork and satisfying higher-level requirements to meet more traditional milestones, checkpoints, or requirements. Depending on how rigid or extensive the requirements are, this can either be quite detrimental to the adoption of Agile or dovetail neatly, without too much effort or distraction.

If the team is engaging in a mini-waterfall process, that might be more of a red flag. This typically happens if the functional areas work separately and do not engage in a high level of communication. There might be a lack of transparency and insight into what each is doing, and handoffs of deliverables might still be a bit formal. The team might not be letting go of old habits and needs more support in adopting new ones. One contributing factor to this might be the UX team, if it is working in parallel sprints, although not every UX team that works in parallel contributes to a mini-waterfall dynamic. If your team is engaging in parallel sprint work and things start looking and feeling awfully like a waterfall,

then it might be worth revisiting how the work is going and if attention needs to be paid to any communication issues.

Jeff Patton

While this is not exactly an Agile term, he is certainly an Agile institution. Many of the UX practitioners whose case studies are in this book received training and coaching from him directly or via webinars. He is a prominent thought leader on Agile and one of the most vocal in speaking about the design process (and the design team) as part of that process. His work and his blog can be found online at www.agileproductdesign.com/. Since most discussion about Agile UX began with him, his work would be the best place to start if you are beginning to explore Agile and UX design.

CASE STUDY—JEFF GOTHELF, THELADDERS.COM

One of the best ways to learn about a process is to hear how other people are doing it. UX teams have been adopting and adapting Agile practices for years and many teams achieved quite a bit of success in working this way. We examine a variety of real teams to see how they managed to implement an Agile process and what lessons they have to offer. Jeff Gothelf's story is interesting for many reasons. It gives a glimpse into an expert realization of Lean UX, which is one of the most recent innovations in Agile and UX. It also paints a picture of the challenges that teams face when making the transition away from traditional methods. We see how getting it wrong once or twice is just a step on the path to getting it right.

At the time of our interview, Jeff was the director of user experience at TheLadders.com, an online subscription job listing service targeted at filling senior-level positions. The organization decided to switch from a waterfall environment to an Agile process and deferred to the UX team to sort out what that would mean for it. Jeff did a lot of work figuring out how to help his team adapt to a wholesale change, and the transition had a rocky start. However, the team members were able to refactor their approach and hit their stride as a team.

Jeff did some research online to see how various UX teams tackle Agile. Since it was a bit early on in the days of Agile UX, few case studies of successful UX teams and an equal amount of horror stories were available online. Jeff started reaching out to people who experienced the transition to get their perspective on working within an Agile environment. Some of his early advisors included Andrew Sandler and Catherine Courage, the Agile UX pioneers at Salesforce.com. He also met Jeff Patton and listened to his take on Agile. The team also had Alistair Cockburn, one of the original authors of the Agile Manifesto, discuss the integration process with them. Essentially, Jeff was able to pick the brains of some of the MVPs of Agile and Agile UX. After getting a sense of the landscape, Jeff put together an idea of the process his team would follow, knowing that, in true Agile fashion, he would revisit the process along the way.

The team started with two-week sprints, working ahead of development and replacing functional specifications with story cards and wireframes. These wireframes started to get quite heavy as annotations were added to document the interaction rules. The team also created a vision document to keep track of the overall plan. Expressing an overall vision can be incredibly helpful for the entire team, not just the UX staff. A common complaint among Agile teams is that they lose track of the big picture, generally because so much of the focus is on the current sprint or immediate piece of work. Having a vision document of some kind, especially one that can be easily remembered, provides a reference point for everyone working on the project. This can go a long way to keeping the day-to-day decisions on the right track, since it is clear to all where they are going.

The first effort at becoming Agile resulted in Jeff being greeted one morning by a diagram created by his team illustrating that everything to do with Agile created a negative environment and low team morale. Jeff describes this as the "pinnacle of unhappiness" for the team in their integration effort. His assessment of what happened was that the UX team did not really know what to expect and did not know what it really meant to be Agile. Additionally, the development team was considered the expert on the process within the organization, but the members had no real idea of where UX should fit in. This situation is actually fairly typical for most UX teams. The UX team is entering new territory, and few, if any, team members have prior experience working with Agile. Regardless of how much training or research the team does, it can be hard to really know what to expect from working in such a new way. Since so much depends on the culture and personalities of the team members involved, every Agile situation is a bit different. Additionally, the interest in and decision to move to an Agile process generally originates from the development organization. If the development team invests in training, it quickly develops expertise, as the members are studying a process that is geared toward their work. Even if the UX team begins its research at the same time, the members have to do the additional homework of trying to figure out the best way to incorporate their work into a predefined process. In the best of situations, the UX team lags behind its development colleagues. This does not need to be a problem, but it can contribute to making the UX team feel like it ts not in control of its own fate.

Since the first approach was less than successful, the UX team members refactored. For their second attempt, they followed the advice of Andrew Sandler, of Salesforce.com, and created a style guide to alleviate some of the design burden and work more efficiently. Using techniques like style guides, style sheets, and pattern libraries can offer predefined solutions. This allows the team to focus on the communication and design of the higher-level elements of the design, rather than spend time and energy explaining how a widget should look or behave. Jeff's team also began prototyping its designs, so that members could communicate their design intentions without having to produce heavyweight documents. This allowed the team to engage in the more complex interaction design problems, rather than trying to do it all in two weeks. All of this put the team more in control of its deliverables with less effort and gave it the ability to more fully engage in the rhythm of Agile. The members also communicated within the UX team about issues as they arose. If a cross-functional issue needed to be dealt with, the directors handled that.

Once the team members were able to execute their work, they started incorporating user testing. They began testing whatever was ready, every other week, mid-sprint. Additionally, two design reviews were added to each of the iterations to get feedback from stakeholders as well as customers. The process continues to evolve, and when I spoke with Jeff, some experienced team members are only sketching ahead then working in parallel with development. Less experienced teams still work ahead but only by a week or so.

In reflecting on the experience, Jeff noted that it is really important for UX team members to be comfortable with opening up their work to critique and collaboration and welcome the interactions with other disciplines. Jeff points out that Lean UX is not the place to try to be a rock star or a hero, and a UX person who is uncomfortable with a team-based culture will not be happy in an Agile environment. This can make the transition even more challenging for some team members. To read more about Jeff's experience getting started with Agile, in his own words, his story can be found at JohnnyHolland.org in the article "Beyond Staggered Sprints: How TheLadders.com Integrated UX into Agile" (Gothelf, n.d.).

What I found most powerful about Jeff's story is that he points out that things did not go well at first. But this does not make the attempt a failure; it is just an effort that needs revisiting and refining. The fact that his team was willing to let him know just how unhappy they were is actually a sign of success that the team culture is such that the members feel that if a problem is raised it will get addressed.

Key Points

- If at first you do not succeed, try again—and again—and again. An Agile process can put a spotlight on pre-existing communication issues and create stress on a UX team, because it introduces a completely new rhythm and way of working. These things take time and effort to overcome, but with time and effort, they can be overcome. The sign of success is not that it is easy but that everyone is communicating and working out problems and wanting to try new solutions.
- Talk to your peers. Many UX teams have gone before you and have lessons to share. The community is quite

generous and Agile veterans are generally very willing to help out anyone who wants it. In addition to the people mentioned in the case study, Jeff also spoke with Jeff Patton and found it valuable to learn from him. Before even speaking with peers, Jeff researched the state of UX and Agile online, so that he could get a sense of challenges. This tactic is a great place to start if you do not even know what questions to ask or who out there might have a good experience to share.

SUMMARY

There are many different ways to apply something called *Agile* to a development process. Regardless of which particular technique is used, it is always important to look to the Agile Manifesto to remember that the spirit of the method is not about adhering blindly to a process or producing copious amounts of documentation. *Being Agile* means working collaboratively as a team and with the customer to produce great software while adapting to change as the need arises. The values and the principles expressed in the Agile Manifesto are at the heart of all the methods, and appreciating them is the first step in being successful with Agile.

References

Cockburn, A., 2005. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, Boston.

Gothelf, J., n.d. Beyond staggered sprints: How TheLadders.com integrated UX into Agile. In: Holland, J., Johnny Holland: It's All About Interaction. Retrieved April 2, 2012, from http://johnnyholland.org/2010/10/beyond-staggered-sprints-how-theladders-com-integrated-ux-into-agile/.

Ladas, C., n.d. Scrum-ban: Lean software engineering. Retrieved April 2, 2012, from http://leansoftwareengineering.com/ksse/scrum-ban/.

Manifesto for Agile Software Development. n.d. Manifesto for Agile Software Development. Retrieved April 2, 2012, from http://www.AgileManifesto.org.

Medlock, M.C., Wixon, D., Terrano, M., Romero, R., Fulton, B., n.d. Using the RITE method to improve products: A definition and a case study (Playtest Research). Retrieved April 2, 2012, from www.microsoft.com/en-us/download/details.aspx?id=20940Twelve.

Patton, J., n.d. Agile Product Design, Holistic Product Design and Agile Software Development. Retrieved April 2, 2012, from http://agileproductdesign.com.

Principles of Agile software. n.d. In: Manifesto for Agile Software Development. Retrieved April 2, 2012, from www.AgileManifesto.org.

Ries, E., 2011. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, New York.

Windows User Experience Design Principles. n.d. :n: MSDN—Explore Windows, Web, Cloud, and Windows Phone Software Development. Retrieved April 12, 2012, from http://msdn.microsoft.com/en-us/library/windows/desktop/dd834141.aspx.

Young, I., 2008. Mental Models: Aligning Design Strategy with Human Behavior. Rosenfeld Media, Brooklyn, NY.