
# SCRUM RISING

## AGILE DEVELOPMENT COULD SAVE YOUR STUDIO

**THREE YEARS AGO, WE AT HIGH MOON STUDIOS WERE IN FULL** crisis. Our first project was slipping. Nothing was working. Customers were wondering where all their money was going. You know, those typical project problems.

As usual, it was the technology effort that was falling behind, and as the new CTO, my job was on the line to manage it. So I started to do some research. This was when I discovered Scrum.

Scrum isn't a big secret. If you read enough project management books, you are bound to run into it sooner than later. Two things struck me about Scrum: 1) it's simple, so you can start using it right away, and 2) it makes common sense. Everything you do in Scrum is done for a clear reason.

In short, Scrum is a methodology that's been used for project management of software development since the early 1990s (its origins are discussed in more detail within this article). With management becoming less of a bottleneck, the talented people we had on staff were able to focus, get on track, and straighten out the project, finishing the game within six months of the original projected date.

Since this experience, I've been talking about Scrum to various developers, and dozens have adopted it. During this time, I've heard a lot of common questions asked, a few myths, and some stories of failures from those coming up to speed on Scrum. This article shares some of these experiences.

### FULFILLING THE SCRUM PROPHECY

Change is painful. The bigger the change or the more people it affects, the more pain. Why would anyone want to make a huge change to an entire company?

For High Moon Studios, we needed to make a change because the alternatives were more painful. We were already suffering from overtime death-marches, and we did not see a working game until very late in our projects. Schedules slipped routinely. We spent too much time putting out too many fires, and our customers were becoming impatient and disappointed.

Kirsten Forbes, a producer who introduced Scrum at Radical Entertainment, says the new methodology allowed her to understand the process of game development in a new and improved way. "At the end of every project, we do a huge

**CLINTON KEITH** *is CTO of High Moon Studios, overseeing research and development of next-generation games, game engines, and agile methods for development. Email him at* ***ckeith@gdmag.com****.*

postmortem. The lament that comes up most often is that the designers change their minds too many times during development," she says. "Well, they didn't change their minds. They learned something about what was fun and adapted the game to accommodate that learning. That's exactly what [designers] should be doing. The problem is not the designers—it's our process. We lock ourselves into a schedule that we can't easily get out of. To make a change in a rigid schedule means we have to find all the tiny areas that those changes ripple through. That's complex and difficult, so it made us averse to change."

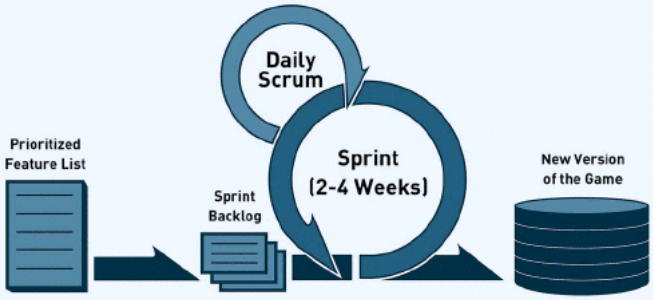


FIGURE 1 In the Scrum cycle, the team creates a new version of the game every two to four weeks.

In sharing the Scrum mindset, I have learned a lot from others who have been struggling with the same problems both within and outside the video game industry. I know one thing for sure: As the challenge of making games grows with the complexity of the platforms and the size of the teams, our methods for how teams work together and how games are developed have to keep up. Just as the technology of the Nintendo 64 doesn't work so well on the Cell processor, the team methodology from the 1980s doesn't work for today's teams.

## WHAT IS SCRUM?

Scrum is an agile methodology, though defining "agile" is a bit difficult.

A few years ago, a number of prominent agile writers got together and tried to define what agile really meant. They came up with what is now called the "agile manifesto," four values that any methodology calling itself "agile" should hold. (See http://agilemanifesto.org for more information.) I've modified it slightly to reflect its use for game developers, and what they need to value:

1. people and communication over processes and project management tools
2. a working game over comprehensive design documents
3. publisher collaboration over milestone definitions
4. responding to change over following a plan.

Because these values are pretty vague, it's hard to translate them into practices. This is where Scrum comes in. Scrum is a set of time-proven practices that translate these values into day-to-day activities.

The term Scrum comes from rugby, specifically the formation in which opposing teams interlock to engage and move the ball up the field. In game development, Scrum has a small team of developers who take ownership of small increments of product development and discuss the product with the customers. The method uses time-boxed iterations of development, called sprints, that create working versions of a game, which allow the customers to see how it is evolving, better understand where the game is going, and have meaningful conversations with the development team.

Scrum teams are small, typically only eight to 10 people. In game development, one team might be dedicated to working on one core mechanic. The team meets daily for 15 minutes, where each member of the team discusses the work they've completed since the last meeting, what they are working on next, and what problems are getting in the way of their progress.

After each iteration review, the team sorts its priorities for the next iteration with the customers, breaks down the tasks, and estimates how long it will take to reach the next iteration. Then they tell the customer what they will deliver.

## FACT VS. FICTION

There are quite a few myths about agile practices and Scrum that come up on a regular basis, the most common of which is that agile development is just another management fad.

"Agile" is a relatively new name for incremental and iterative methods and other project practices that have been around for a long time. The agile methodologists combined a number of these ideas rather than inventing new ones. In fact, if you look back 20 years, you'll find that game development teams were typically so small that they practiced incremental and iterative development out of necessity. Game development was far less expensive and time-consuming then, too, so it was a lot more experimental. Publishers could afford 10 failures for every hit, and finding the hit was more critical than nailing the budget.

Another myth is that there is no overtime on agile projects. Agile teams choose how much work they can commit to every two to four weeks. The benefit is that they own their work and aren't trying to meet a schedule they don't believe in. They commit to a level of work that they feel they can accomplish without overtime. However, there are too many uncertainties in even two weeks of work, and a team can often wind up with more work than they can complete without overtime.

Sometimes an overloaded team compensates for the heavy workload by dropping some of its lower priorities. Iterations cannot be delayed, therefore they free developers to complete the higher priority goals within the time frame originally established. Other times, the team has to put in extra hours to meet a commitment. In either case, it's often left up to the team. A few extra hours a day for the last week of the iteration can make a huge difference in what the team produces in that iteration. The goal isn't to stop work after eight hours—it's to produce the best possible value in a sustainable pace.

A third myth about agile development is that long-term plans do not exist. We spend more time planning during an agile project than we ever did in a waterfall project. The main difference is that it's spread out over the entire project, not just done at the start. We plan for the entire scope of the game up front, but we don't try to be deterministic and plan away uncertainty. We focus on *planning* instead of *the plan*

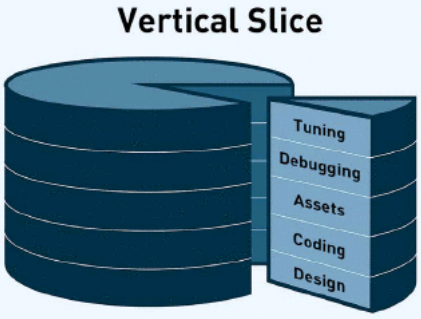


FIGURE 2 **Each feature added in a sprint has a vertical slice of work done for it.**

because we want to be able to embrace change and adjust our forecast for the game based on what we learn.

At High Moon, we set aside a full day every three months to refine and review the full project plan. This allows us to review parts of the game's development, such as asset production, which can't be as fully agile as some of the early work done on core mechanics—for example, we wouldn't want to change our jump heights after we've built half the levels.

Another myth I've heard is that games developed with an agile process will be great. Of course, no methodology can ensure great games. A bad game idea, or worse, a team that lacks talent, will fail regardless of the methodology used. Perhaps the only benefit of agility in these situations is that the game will fail fast. Conversely, agile development can help talented teams avoid many failures that other methodologies allow.

In general, the benefits of working agilely are that it can reduce waste and allow your team and customers to make better choices along the way. The people in your company are your greatest asset; having a healthy company culture is important. Because agile development focuses on communication, ownership, and commitment, a company's culture can really improve as a result of adopting an agile approach.

One final myth is that leads are not necessary on agile game projects. This particular myth was one we fell for when we first started using Scrum. Scrum eliminates many of the tasks required of a lead because teams become self-organizing. Teams break down to estimate and track their tasks at a team level, relieving the leads from doing these sorts of jobs, which they usually dislike in the first place.

The idea of taking one of your most talented programmers and artists and instructing them to estimate and track tasks for half their time is not a good idea—yet, we do it all the time in our industry. However, even an agile team still needs leadership. The agile team lead focuses on mentoring and communicating with the other leads about the needs of a project, ensuring that all the teams are focused on producing a common product together.

## OPENING THE SCRUM GATE

Game studios that are interesting in adopting Scrum should start small. Don't try to convert the entire studio overnight; instead, identify a team of approximately eight developers (no more than 10) who will be the Guinea pigs for this experiment. The team should be willing to try out Scrum and be objective about the results.

Next, identify the champion for Scrum, ideally someone who is a lead or producer from the Guinea pig team. This is going to be the Scrum expert who learns all about Scrum. Once a champion is chosen, have him or her read two essential books on Scrum by Ken Schwaber (see Resources, page 26).

The most important, yet also most controversial piece of advice is that you start Scrum by the book with the first team, following every practice that exists, even if you don't fully understand the value.

The practices of Scrum are derived from very important principles, including ownership, accountability, transparency, and teamwork, which amount to the central benefits of using Scrum to make games. If you change some of the practices without understanding the underlying philosophy, you'll never see many of those benefits. As Schwaber says, "Every time I've seen someone need to get rid of one of Scrum's basic mechanisms or rules, it is because the mechanism or rule is making something visible that nobody wants to see. So, they get rid of the rule and the problem becomes invisible. For example, 'We don't need daily scrums, so let's only have them every week.' The daily Scrum is making visible that the team isn't self-managing and doesn't work as a team, but as a group of individuals doing their own things. No need for information to be interchanged because nobody cares."

REGULARLY SCHEDULED STATUS UPDATE MEETINGS WILL GO AWAY. YOUR PROJECT MANAGERS AND PRODUCERS WILL NOT BE INTERRUPTING YOUR WORK ALL THE TIME TO ASK FOR A GLIMPSE OF YOUR PROGRESS.

This isn't to say that the Scrum practices should never be changed—they are meant to be changed by the teams to improve their velocity. I also suggest that teams who are freshly adopting a Scrum methodology attend a Scrum Master Certification Course. It's a two-day course taught at many locations and times (see Resources).

Another important step new Scrum sign-ons need to take is to identify the customers for the team and define a few goals for that team to accomplish in a prioritized list. The team must then pick someone to be the Certified Scrum Master who will explain the reasons behind every step of development to the customers and the team the first time around.

Finally, the team is ready to begin the sprint cycle, the core development phase of Scrum (see Figure 1). Sprint cycles include: sprint planning, daily Scrum meetings, and sprint reviews. It's outside the scope of this article to describe these practices in detail, but plenty of information is available in the Resources.


**In the daily Scrum meeting, each team meets for 15 minutes to go over everyone's tasks and identify problems.**

## OBSTACLES TO INITIATION

*1. Team buy-in.* Kirsten Forbes says she faced an initial difficulty in getting the Radical team to accept a Scrum way of life. "When I presented [Scrum] to the entire team, I walked through the same process but added a few spicy advantages to encourage them," she says. "For team members, the benefits are clear. Regularly scheduled status update meetings will go away. Your project managers and producers will not be interrupting your work all the time to ask for a glimpse of your progress. You control how your work gets done because you break the backlog down into tasks. And you do your own time estimates and you learn to get it right."

*2. Management buy-in.* It's easy to oversell Scrum to management and ruin things for yourself. Working with one team to evaluate Scrum for a few months is an easier sell than asking management to convert the entire studio overnight. Once the higher-ups see the level of performance from the Scrum team, the solution will sell itself.

*3. Getting used to iteration.* It takes a few months for the team to get used to the pacing of Scrum sprints. The first impulse a team usually shows is to treat sprints like mini waterfalls with small design documents written at the start and working code not coming together at the end. This is still better than traditional waterfall, but coaching the team to keep them communicating with each other and keep the build working (instead of writing documents) will improve performance.

*4. Code iteration: how to avoid spaghetti.* When the goals of a project can change every sprint, it's hard for the code base to keep pace and not slow the programmers down. Setting aside time for the team to refactor the code on a regular basis is valuable. Programmers might want to investigate some of the practices of Extreme Programming, such as Test-Driven Development—a very useful way of creating code that can keep pace in an agile environment.

*5. Publisher buy-in.* As with management, publishers asked to accept Scrum should be approached in a very low-key way. The initial hurdle for publishers is often the idea of having a more flexible milestone delivery list. Developers have found that if they include the publisher as one of the customers at every sprint review (either by having them visit or through conference calls while they play the game), it's a very beneficial selling point for Scrum. Once they see how their feedback is considered and possibly included in the next sprint review, they will become more enthusiastic. Publishers are well aware that the traditional milestone-based contracts create a relationship in which change is resisted, so they are likely to see the benefit of Scrum firsthand.

## VERTICAL SLICES AND TEAM SIZES

One of the main differences between Scrum and the waterfall method is the idea that the product is kept at a state of near-completion every sprint, and that features added every sprint

## RESOURCES

Agile Manifesto
**http://agilemanifesto.org**

Suggested reading, mailing list, and more from the author
**www.agilegamedevelopment.com**

Nonaka, Ikujiro and Takeuchi, Hirotaka. *The Knowledge-Creating Company.* New York: Oxford University Press, 1995.

Schwaber, Ken. *Agile Project Management with Scrum.* Redmond, Wash.: Microsoft Press, 2004.

Schwaber, Ken and Beedle, Mike. *Agile Software Development with Scrum.* Upper Saddle River, N.J.: Prentice Hall, 2001.

Ken Schwaber's site, with Scrum Master Class schedules
**www.controlchaos.com**

have a level of completeness that improves the value of the final game. The goal is to have the value of the feature proven during every sprint. Design, coding, debugging, tuning, and assets are all taken into consideration.

A common misconception is that it becomes impossible to implement any work that takes longer than one sprint. That's just not true. The purpose of working this way is to show the customers the value of the feature every two to four weeks, and to show how it improves sprint-by-sprint.

In planning the work for each feature, the team considers a vertical slice (see Figure 2). Think of a vertical slice as a slice of five layer cake: The bottom layer is design, followed by coding, assets, debugging, and tuning. With waterfall, the team would create the entire bottom layer first, leaving the frosting until the end of the project, if there's time left at all. With Scrum, the cake is built one complete slice at a time instead of layer by layer.

Defining what a vertical slice is per feature is not easy. For example, if a team is working on AI, does it need characters with final models and animations to prove that AI characters are fun? Does it want to find out how many characters should be used in a scene before figuring out what the character asset budgets are? What about the trade-offs between character quantity and quality? How does anyone know when it's done?

"'Done' means fully tested and integrated at each sprint," says Forbes, citing Radical's definition. "Early on, we found some items that one team member considered done were contentious. Not everyone agreed it was done. To resolve this, we added a column on our task board for verified," she says. "The customer receiving the piece of work verifies it.This would be the artists if a tool fix is made for them, or the art director if a piece of art is completed, for example. The Scrum master is responsible for making sure everything has been properly verified before it comes off the task board."

There are no hard and fast rules on this. A typical example would be the trade-off between the quality versus quantity of characters you want in a game. If the quantity of characters is more important to the customers, they would want the team to focus on developing the AI for many characters and defer the quality decision a bit longer. Many times this decision is made too early, too late, or based on what the customers might not want. We want to avoid the situation where we have to throw out the work we have done because it doesn't fit the budget, or chop out key features just to ship the game.

Teams building next-generation games can exceed 70 people in size. With Scrum, no group should exceed 10 people. What results is seven teams focusing on different areas of one game, which can create dependency and communication problems.

A common reaction is for management to jump in and solve problems for the team, but the right thing to do is coach the teams to self organize and lead themselves out of their problems. Scrum methodology allows for this, but it's sometimes difficult for management and team members to let go and allow the teams themselves to take ownership.

## SCRUMPTIOUS RESULTS

I can't guarantee that Scrum is the solution for all project management problems. All I can claim is that it's working well for us at High Moon. It's not as simple as reading a book and applying the practices. It's easy to start, but it takes a long time to truly understand the ideology behind it. Since the road to adoption is different for every studio, it's best to share experiences with a wide range of other adopters.

## a brief history of scrum

**THE TERM SCRUM ORIGINATES FROM** an article that appeared in the January-February 1986 issue of *Harvard Business Review* written by Japanese business researchers Hirotaka Takeuchi and Ikujiro Nonaka. Titled "The New New Product Development Game," the article explored companies in the U.S. and Japan that were delivering innovative products very rapidly.

The researchers found that these companies, all of whom were developing highly sophisticated products such as automobiles and consumer electronics that required a lot of complexity in terms of design and manufacturing, shared a number of characteristics in their product development processes. Among these were self-organized teams that were cross-functional and the development of products on an iterative basis. The researchers likened the behavior to what happens in the sport of rugby, as mentioned in the article.

Jeff Sutherland and Ken Schwaber began formulating the Scrum software development process in the early 1990s, and announced it in 1995. Their variation on the methodology adopted common good practices that had existed for many years into a framework for teams that self-organized and communicated and cooperated closely to develop software in an iterative and incremental way.

Schwaber and Sutherland formalized their philosophy throughout the 1990s, and in 2001 they gathered with other groups to create the "Agile Manifesto" (see Resources), which defines the values of every agile methodology, including Scrum. Today, Schwaber is considered the leading consultant on certified Scrum Master training.

Over the past five years, use of Scrum and Extreme Programming has grown tremendously, driven in part by studies that have shown as much as six-fold improvement in time and cost to create new products. Technology innovators Google, Yahoo, and Microsoft have adopted it, as have companies in traditionally more conservative industries such as banking and insurance.

High Moon adopted Scrum in 2004 in the last year of development of our debut game DARKWATCH, followed by our implementation of Extreme Programming the subsequent year. Since then we have seen these methodologies spread throughout the game industry, as dozens of developers have chosen to implement both methods.