The Addison-Wesley Signature Series

A Mike Cohn Signature Book

Agile Game Development with Scrum

Clinton Keith

Foreword by Mike Cohn

Library of Congress Cataloging-in-Publication Data

Keith, Clinton.
Agile game development with Scrum / Clinton Keith.
p. cm.
Includes index.
ISBN 0-321-61852-1 (pbk. : alk. paper) 1. Computer games—Programming. 2. Agile software development. 3. Scrum (Computer software development) I. Title.
QA76.76.C672K45 2010
005.1—dc22

2010006513

ISBN-13: 978-0-321-61852-8
ISBN-10: 0-321-61852-1
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2010

Foreword

The insight that Scrum (indeed, agile software development in general) and game development were a near-perfect match was no surprise to Clinton Keith. As the CTO of his studio, he was a pioneer in the pairing of Scrum and game development. Though some were skeptical, Clint saw the possibilities, and as a result, he not only created the first game developed using Scrum but also helped his teams put the fun back into game development.

And why shouldn't game development be fun as well as profitable? It's true that the game industry is well known for aggressive deadlines and that teams are working with ambiguous requirements in a very fluid marketplace, but that is exactly the kind of environment where Scrum can help the most. Because Scrum is iterative and incremental and forces a team to put the game into a playable state at least every two to four weeks, the team members can see new features and scenarios develop right before their eyes.

In *Agile Game Development with Scrum*, Clint shares his experience and insights with us. He tells us everything we need to know to successfully use Scrum in the challenging field of game development. In doing so, he provides an introduction to agile and Scrum and tells us how they can help manage the increasing complexity facing most game development efforts. He explains how something as large and integrated as "AAA" console games can be developed incrementally. Along the way, Clint offers invaluable guidance on getting all of the specialists who are necessary on a game project to work together in an agile manner. He even delves into how to use Scrum when working with a publisher. In providing all of this guidance, Clint doesn't shy away from the challenges. Instead, he generously shares his advice so that we can perhaps avoid some of them.

There is little doubt in my mind that the book you are holding can have a profound effect on any game project and studio. Once introduced to and accustomed to Scrum, team members will not want to work any other way. They will have learned what Clint knew long ago—that Scrum is the best way to handle the complexity and uncertainty of game development.

—Mike Cohn
Cofounder, Scrum Alliance
and Agile Alliance

Preface

This book was written for game developers who either are using agile methodologies or are curious about what it all means. It condenses much information from a number of fields of agile product development and applies it to the game industry's unique ecosystem. It's based on the experiences of dozens of studios that have shipped games using agile over the past six years.

If you are not in the game industry but curious about it or agile, you should enjoy this book. Since the book needs to communicate to every discipline, it doesn't get bogged down in the specifics of any one of them because, for example, artists need to understand the challenges and solutions faced by programmers for cross-discipline teams to work well.

As you can tell from the title, this book focuses on Scrum more than any other area of agile. Scrum is a discipline-agnostic framework to build an agile game development process. It doesn't have any defined art, design, or programming practices. It's a foundation that allows you and your teams to inspect every aspect of how you make games and adapt practices to do what works best.

How did agile and game development meet? For me, it started in 2002 at Sammy Studios. Like many studios, our path to agile came by way of impending disaster. Sammy Studios was founded in 2002 by a Japanese Pachinko manufacturing company. Their goal was to rapidly establish a dominant presence in the Western game industry. To that end, Sammy Studios was funded and authorized to do whatever was needed to achieve that goal.

As seasoned project managers, we quickly established a project management structure that included a license of Microsoft Project Server to help us manage all the necessary details for our flagship game project called Darkwatch.

The plan for Darkwatch was ambitious. It was meant to rival Halo as the preeminent first-person console shooter. At the time, we thought that as long as we had the resources and planning software, little could go wrong that we couldn't manage.

It didn't take long for many things to go wrong. Within a year we were six months behind schedule and slipping further every day. How was this happening?

- **Disciplines were working on separate plans:** Each discipline had goals that permitted them to work separately much of the time. For example, the animation technology was being developed according to a plan that called for many unique features to be developed before any were proven. This resulted in the animation programmer working on limbs that could be severed while the animators were still trying to make simple transitions work. Correcting these problems required major overhauls of the schedule on a regular basis.

- **The build was always broken:** It took exceptional effort to get the latest version of the game working. The Electronic Entertainment Expo (E3) demos took more than a month of debugging and hacking to produce a build that was acceptable. Even then, the game had to be run by a developer who had to frequently reboot the demo machine.
- **Estimates and schedules were always too optimistic:** Every scheduled item, from small tasks to major milestone deliverables, seemed to be late. Unanticipated work was either completed on personal time or put off for the future. This led to many nights and weekends of overtime work.
- **Management was constantly "putting out fires" and never had time to address the larger picture:** We managers selected one of the many problems to fix each week and organized large meetings that lasted most of a day in an attempt to solve it. Our list of problems grew faster than our ability to solve them. We never had the time to look to the future and guide the project.

The list goes on, and the problems continued to grow. Most problems were caused by our inability to foresee many of the project details necessary to justify our comprehensive plan's assumptions beyond even a month. The bottom line was that our planning methodology was wrong.

Eventually our Japanese parent company interceded with major staff changes. The message was clear: Since management was given every possible resource we wanted, any problems were our own fault, and we were given short notice to correct them. Not only our jobs but also the existence of the studio hung in the balance.

It was in these desperate times that I began researching alternative project management methods. Agile practices such as Scrum and Extreme Programming (XP) were not unknown to us. The original CTO of Sammy had us try XP, and a project lead was experimenting with some Scrum practices. After reading a book about Scrum (Schwaber and Beedle 2002), I became convinced that it could be used in our environment.

Upon discovering Scrum, we felt that we had found a framework to leverage the talent and passion of game development teams. It was challenging. The rules of Scrum were biased toward teams of programmers creating IT projects. Some things didn't work.

This began an endless series of discoveries about what *agile* meant and what worked for game developers. I began speaking about agile game development in 2005. This was around the time that studios were developing titles for Xbox 360 and PlayStation 3. Teams of more than 100 people were becoming the norm, and project failures cost in the tens of millions. Unfortunately, many took the agile message too far and perceived it as a silver bullet for some.

In 2008, after speaking with hundreds of developers at dozens of studios, I decided that I enjoyed helping game developers adopt agile enough to become a full-time independent coach. I now coach many studio teams a year and teach developers how to be ScrumMasters in public classes. My experiences working with and learning from these developers have led to this book.

Organization

Part I, "The Problem and the Solution," begins with the history of the game industry. How have the industry's products and methodologies for development changed? What has led us to bloated budgets, schedules that are never met, and project overtime death marches? It concludes with an overview of agile and how the problems of managing the development of games can benefit from agile's values.

Part II, "Scrum and Agile Planning," describes Scrum, its roles and practices, and how it's applied to game development. It describes how a game's vision, features, and progress are communicated, planned, and iterated over the short and long term.

Part III, "Agile Game Development," describes how agile is used over the full course of a game development project, including where some of the Scrum practices can be supplemented with lean principles and kanban practices for production. It explores agile teams and how Scrum can be scaled to large staffs, which might be distributed across the globe. Part III concludes by examining how teams continuously improve their velocity by decreasing the time required to iterate on every aspect of building a game.

Part IV, "Agile Disciplines," explains how each of the widely diverse disciplines work together on an agile team. It describes the role of leadership for each discipline and how each one maps to Scrum roles.

Part V, "Getting Started," details the challenges and solutions of introducing agile practices to your studio and publisher. Overcoming cultural inertia and integrating agile principles into a studio's unique processes—without destroying the benefits—can take time, and there many challenges along the way. The chapters in this part are a guide to meeting these challenges.

Although this is a starting place for agile game development, it is by no means the end. There are great books about Scrum, Extreme Programming, lean, kanban, user stories, agile planning, and game development. These books will provide all the detail desired on the path of continual improvement.

Developers for iPhone, PC, and massively multiplayer online games use the practices described here. I share many stories based on my technical background, and indeed there are more existing practices for the agile programmer, but the book applies to the entire industry. There are stories and experiences shared from many people from every discipline, genre, and platform.

Chapter 12
Agile Design

When I first started working on games professionally in the early nineties, the role of designer was being instituted throughout the industry. Following the mold of prominent designers such as Shigeru Miyamoto and Sid Meier, designers were seen as directors of the game, or at least the people who came up with many of the ideas. The role required communication with the team on a daily basis but not much written documentation.

As technical complexity, team size, and project durations grew, the role of the designer became more delineated. Some projects had teams of designers who specialized in writing stories, scripting, tuning characters, or creating audio. Hierarchies emerged to include lead, senior, associate, or assistant designers, among others.

The overhead of communication with large teams and the cost of longer development efforts led to a demand for certainty from the stakeholders. Large detailed design documents attempted to create that certainty, but at best they only deferred its reckoning.

This chapter examines how agile can help reverse this trend.

VIEWPOINT

"Designers are the chief proponents for the player. This has not changed in 20 years of game development. Though titles and roles have changed, designers look out for gameplay and quality of the product from a player's perspective.

"When teams were small—with ten or less people—this could be done easily; it was a series of conversations while textures were created and code was written. The design was natural and organic as it emerged from the team. 'Horse swaps' could easily occur. For example, trading a very difficult-to-build mechanic for an easy one that still achieved the same gameplay vision was relatively simple.

"However, in the past ten years, teams have begun to balloon, first to the 30- to 50-person teams of the nineties and then finally to the occasional several-hundred-person monstrosities of the 2000s. A single designer could not have all the conversations that needed to happen (even several designers have problems). As a result, documentation began to surface that outlined the product as a whole, from the very high level to the very granular. Although this paints the initial vision of the title, it does away with one of the most important facets of any type of product development: the dialogue.

"Scrum addresses this. Five- to ten-person cross-discipline Scrum teams usually include a designer. Each of these designers is entrusted by the lead designer to understand the key vision elements and speak to the team."

—Rory McGuire, game designer

The Problems

What are some of the problems that face developers on large projects? The two most common problems are the creation of large documents at the start of a project and the rush at the end of the project to cobble something together to ship.

Designs Do Not Create Knowledge

Originally when designers were asked to write design documents, they rebelled. Writing a design document seemed like an exercise to placate a publisher or commit the designers to decisions they weren't ready to make. Over time this attitude toward documentation has changed. Writing design documents has become the focus for many designers. It's felt that this is the easiest way to communicate vision to both the stakeholders and a large project team.

Designers need to create a vision, but design documents can go too far beyond this and speculate instead. Once, on a fantasy shooter game I worked on, the designers not only defined all the weapons in the design document but how many clips the player could hold and how many bullets each clip contained! This level of detail didn't help the team. In fact, for a while, it led them in the wrong direction.

The Game Emerges at the End

At the end of a typical game project, when all the features are being integrated, optimized, and debugged, life becomes complicated for the designer. This is the first time they experience a potentially shippable version of the game. At this point it typically bears little resemblance to what was defined in the design document, but it's too late to dwell on that. Marketing and QA staffs are ramping up, and disc production and marketing campaigns are scheduled.

The true performance of the technology begins to emerge, and it's usually less than what was planned for during production. This requires that budgets be slashed. For example, waves of enemy characters become trickles, detailed textures are decimated, and props are thinned out.

Because of deadlines, key features that are "at 90%" are cut regardless of their value. As a result, the game that emerges at beta is a shadow of what was speculated in the design document. However, it's time to polish what remains for shipping.

Designing with Scrum

Successful designers collaborate across all disciplines. If an asset doesn't match the needs of a mechanic, they work with an artist to resolve the issue. If a tuning parameter does not exist, they work with a programmer to add it. They also accept that design ideas come from every member of the team at any time. This doesn't mean that every idea is valid. The designer is responsible for a consistent design vision, which requires them to filter or adapt these ideas.

COPS AND ROBBERS

In the late nineties, while we were developing Midtown Madness, I was playing "capture the flag" after-hours in the game Team Fortress. One day it occurred to me that a version of "capture the flag" for our city racing game might be fun. I raised this idea with the game designer, and he suggested a creative variation called "cops and robbers." In it, one group of players are robbers, while the other group are cops. The robbers try to capture gold from a bank and race to return it to their hideout. The cops try to stop the robbers and return the gold. This feature was a big hit with online players and seemed to be even more popular than racing! Good ideas can come from anywhere!

A Designer for Every Team?

A designer should be part of every cross-discipline Scrum team working on a core gameplay mechanic. They should be selected on the basis of the mechanic and their skills. For example, a senior designer should be part of the team working on the shooting mechanic for a first-person shooter. If the team is responsible for the heads-up display (HUD), then a designer with a good sense of usability should join the team.

The Role of Documentation

When designers first start using Scrum, they'll often approach a sprint as a mini-waterfall project; they'll spend a quarter of the sprint creating a written plan for the work to be done during the remainder. Over time this behavior shifts to daily collaboration and conversation about the emerging goal. This is far more effective.

This doesn't mean that designers shouldn't think beyond a sprint and never write any documentation. A design document should limit itself to what is known about the game and identify, but not attempt to answer, the unknown. Documenting a design forces a designer to think through their vision before presenting it to the rest of the team. However, a working game is the best way to address the unknown.

A goal of a design document is to share the vision about the game with the team and stakeholders. Relying solely on a document for sharing vision has a number of weaknesses:

- **Documents aren't the best form of communication:** Much of the information between an author and reader is lost. Sometimes I've discovered that stakeholders don't read any documentation; it's merely a deliverable to be checked off!
- **Vision changes over time:** Documents are poor databases of change. Don't expect team members to revisit the design document to find modifications. Recall the story of the animal requirement for Smuggler's Run; that was a case of failed communication about changing vision.

Daily conversation, meaningful sprint and release planning, and reviews are all places to share vision. Finding the balance between design documentation and conversation and collaboration is the challenge for every designer on an agile team.

"STAY THE %#&$ OUT!"

One designer at High Moon Studios had a difficult time shifting his focus away from documentation when he joined his first Scrum team. At the start of every four-week sprint, he locked himself in an office for a week to write documentation for the sprint goal. The team didn't want to wait and pestered him with questions during this time. The constant interruptions led the designer to post a note on his door that read "Stay the %#&$ out! I'm writing documents!" Eventually, the team performed an "intervention" of sorts with the designer to get him to kick the documentation habit!

Parts on the Garage Floor

Agile planning practices create a prioritized feature backlog that can be revised as the game emerges. The value of features added is evaluated every sprint. However, many core mechanics take more than a single sprint to demonstrate minimum marketable value. As a result, the team and product owner need a certain measure of faith that the vision for such mechanics will prove itself. However, too much faith invested in a vision will lead teams down long, uncertain paths, which results in a pile of functional "parts" that don't mesh well together. I call this the "parts on the garage floor" dysfunction.

We saw one such problem on a project called Bourne Conspiracy. In this third-person action-adventure game, the player had to occasionally prowl around areas populated with guards who raise an alarm if they spot the player. This usually resulted in the player being killed. In these areas, the designers placed doors that the player had to open. At one point, a user story in the product backlog read as follows:

> As a player, I want the ability to pick locks to get through locked doors.

This is a well-constructed story. The problem was that there were no locked doors anywhere. This resulted in another story being created:

> As a level designer, I want to have the ability to make doors locked so the player can't use them without picking the lock.

This story is a little suspect. It represents value to a developer, but it doesn't communicate any ultimate value to the player. Such stories are common, but they can be a symptom of a debt of parts building up.

The parts continued to accumulate as sprints went by:

> As a player, I want to see a countdown timer on the HUD that represents how much time is remaining until the lock is picked.

> As a player, I want to hear lock-picking sounds while I am picking the lock.

> As a player, I want to see lock-picking animations on my character while I pick the lock.

This continued sprint after sprint; work was being added to the lock-picking mechanic. It was looking more polished every review.

All of these lock-picking stories were building the parts for a mechanic that was still months away from proving itself. The problem was that lock picking made no sense. The player had no choice but to pick the locks. Nothing in the game required the player to choose between picking a lock or taking a longer route. Ultimately, the vision was proven wrong, and lock picking was all but dropped from the game despite all the work dedicated to it.

Figure 12.1 illustrates this problem of "parts on the garage floor."

The figure shows many parts, developed over three sprints, finally coming together in the fourth. This represents a debt that could waste a lot of work if it doesn't pay off. It also prevents multiple iterations on the mechanic over a release cycle, because the parts are integrated only in the last sprint.

Ideally, each sprint iterates on a mechanic's value. Figure 12.2 shows the parts being integrated into a playable mechanic every sprint or two.

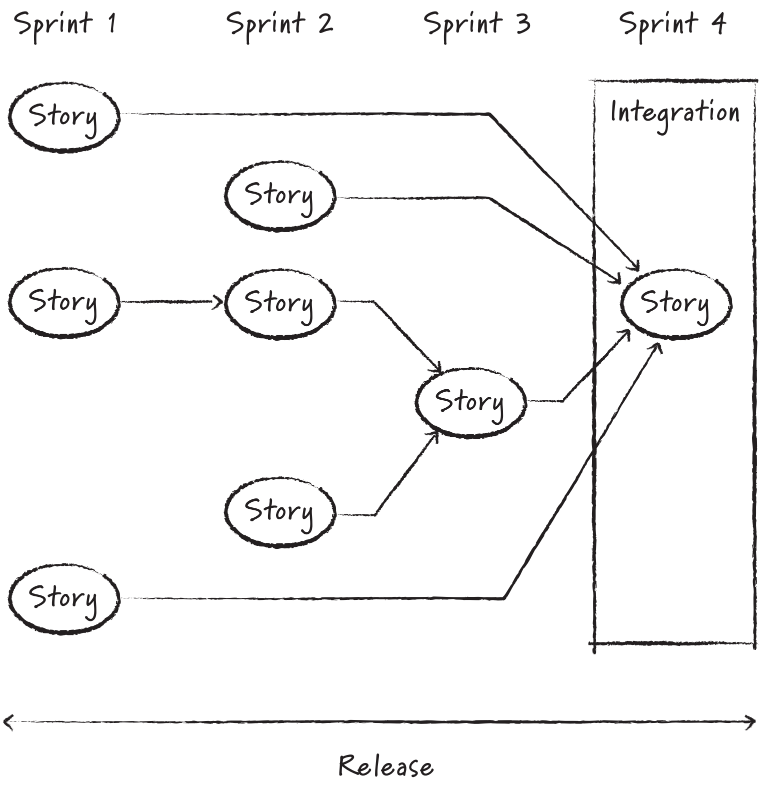

FIGURE 12.1 Integrating a mechanic at the end of a release

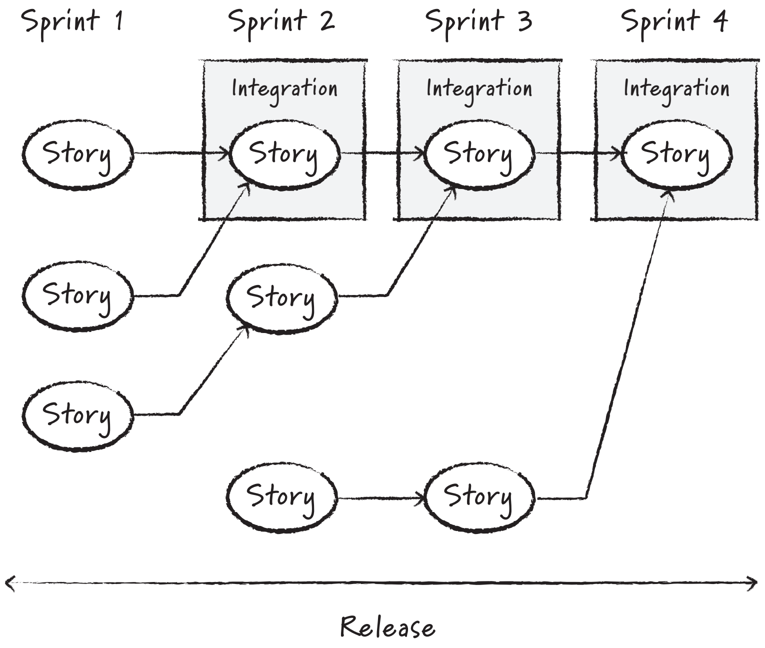

FIGURE 12.2 Integrating a mechanic every sprint

The approach changes the stories on the product backlog:

> As a designer, I want doors to have a delay before they open. These doors would delay the player by a tunable amount of time to simulate picking a lock while the danger of being seen increases.

Notice that this story expresses some fundamental value to the player, which communicates a vision to both stakeholders and developers.

> As a designer, I want to have guards simulating patrols past the locked doors on a regular basis so the timing opportunity for the player to pick the lock is narrow.

> As a player, I want to unlock doors in the time that exists between patrols of armed guards to gain access to areas I need to go.

The first few stories are infrastructure stories, but they describe where the game is headed. They build the experience for the player in increments and explain why. The value emerges quickly and enables the product backlog to be adapted to maximize value going forward. This is in stark contrast to building parts that assume a distant destination is the best one. Iterating against a fixed plan is not agile.

CREATING FUN IS ITERATIVE AND COLLABORATIVE BY NATURE

One year I took my family to Colorado to spend Christmas in a cabin. After a large snowstorm, my sons wanted to sled on the side of a small hill. So, I went to the local hardware store but could only find a couple of cheap plastic sleds. At first, the snow was too thick and the hill was too small for the sleds, so we packed down a path in the snow and built a starting ramp for speed. The sleds kept running off the track, so we packed snow on the sides. To increase speed, we poured water on the track to ice it—it began to look like a luge track!

After a few hours we had a great track. The boys would speed down on their sleds. They built jumps and curves and even a few branches into the track.

My oldest son said, "It's lucky that you bought the perfect sleds!" I hadn't done that, so we talked about it. The sleds weren't perfect; we had merely iterated on the track to match their characteristics. We added elements, such as the sides to the track, to overcome the sled's lack of control. We added other features, such as the ramp and track ice, to overcome the limitations of the thick snow and low hill. The sleds were the only thing that couldn't be changed.

I couldn't help comparing this to game development. We created an experience by iterating on things we had control over and adapted for things we didn't. In this case, design was entirely constrained to working with the level (the track) and not the player control (the sled), and we were still able to "find the fun"!

Set-Based Design

When a project begins, the game we imagine is astounding. Players will experience amazing gameplay and explore incredible worlds where every turn reveals a delightful surprise. However, as we develop the game, we start to compromise. Imagination hits the limits of technology, cost, skill, and time. It forces us to make painful decisions. This is a necessary part of creating any product.

Identifying and narrowing down the set of possibilities is part of planning. For example, when we plan to create a real-time strategy game, we eliminate many of the features seen in other genres from consideration (see Figure 12.3).

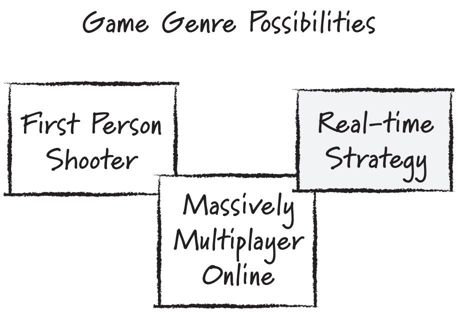

FIGURE 12.3 Narrowing the game to a specific genre

Planning continues to narrow down the set of possible features. Following a high-level design, many developers refine discipline-centric designs. Designers plan the game design possibilities, programmers plan the technical design possibilities, and artists plan the art design possibilities. These possibilities do not perfectly overlap. For example, the designers may want large cities full of thousands of people, but the technology budget may only allow a dozen characters in linear levels. Figure 12.4 shows how the union of design, art, and technical possibilities overlap to create a set of features that all disciplines agree upon.

As mentioned earlier, the project starts with an area quite large in scope. As time goes by, the project staff gains more knowledge of what is possible, and the range of possibilities shrink, as shown in Figure 12.5.

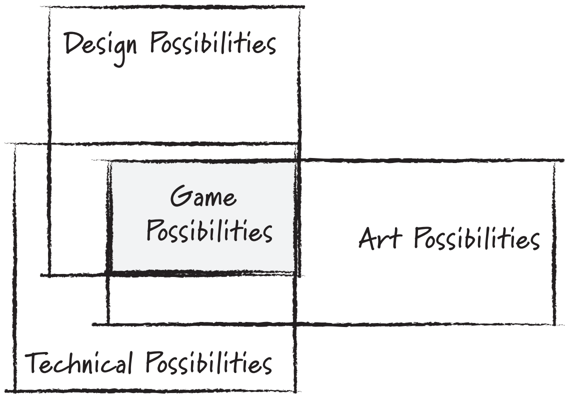

FIGURE 12.4 The set of possibilities at the start of a project

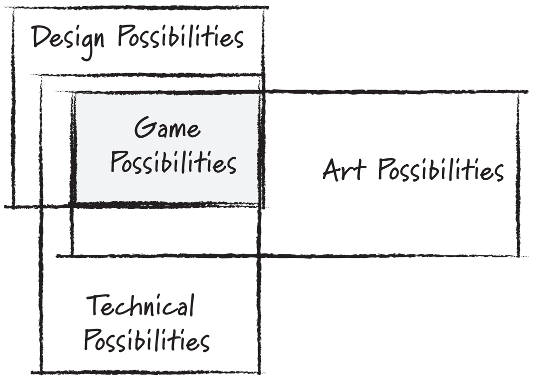

FIGURE 12.5 The set of possibilities as the project progresses

This refinement of scope slowly happens through iteration and discovery. It requires cross-discipline collaboration to find a common ground so that effort is spent on a rich set of features possible for everyone to succeed.

Problems occur when the disciplines branch off from one another and plan in isolation. If the disciplines refine their set of possibilities too early or in isolation, then it greatly reduces the set of overlapping options for the game. This approach is called **point-based design** in which a single discipline design is refined in isolation (usually the game design). The set of design options have been narrowed so much that the overlapping game feature set has been vastly reduced, as shown in Figure 12.6.

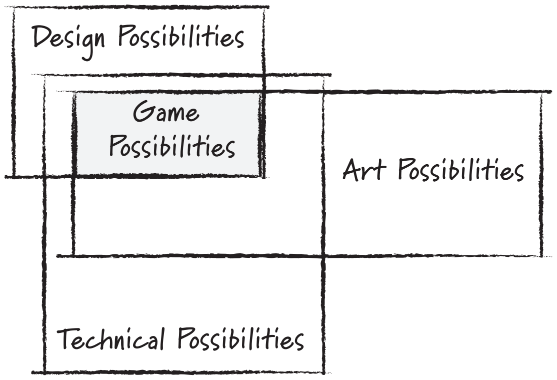

FIGURE 12.6 Narrowing game design too soon

This is the reason for cross-discipline planning. It keeps options open and the union of all sets as large as possible, so when more is learned, the project has a wider range of options.

An example of the problem with a point-based design was with a level-streaming decision made early on a game called Darkwatch. Early in development the designers decided that contiguous sections of levels had to be streamed off the game disc in the background during gameplay so that the player felt the game was taking place in one large world. The decision was made although no technical or art tool solutions for such streaming existed.

Entire level designs were created based on the assumption that the technology and tool set would be created and that artists would be able to author the streaming levels efficiently. Unfortunately, these assumptions were proven false. The effort required to implement the full streaming technology left no time to create the tools necessary for the artists to manipulate the levels. As a result, the levels were "chopped up" into small segments, and these segments were loaded while the player waited. The gameplay experience suffered greatly from this.

Another approach to narrowing multidiscipline designs, called **set-based design**, is used to keep design options alive as a number of solutions are explored and the best design is converged upon. Set-based design has been shown to produce the best solutions in the shortest possible time (Poppendieck and Poppendieck 2003).

A set-based design approach to such a problem as the streaming level example is different from a typical point-based design. Instead, a number of options are explored:

- A full level-streaming solution
- A solution that streams portions of the levels (props and textures)
- No streaming at all

As each option matures, knowledge is built to enable a better decision to be made before level production. Potential solutions are dropped as soon as enough is learned about cost, risk, and value to show that they weren't viable. Although developing three solutions sounds more expensive, it is the best way to reduce cost over the course of the project.

Making decisions too early is a source of many costly mistakes. This is difficult to combat since such decisions are often equated with reducing risk or uncertainty. In point of fact, early decisions do not reduce risk or uncertainty. The delay of the level design decision in the set-based design approach is an example of postponing a decision as long as it can be delayed and no longer. This is an essential part of set-based design.

Lead Designer Role

The lead designer's role is similar to other lead roles; they mentor less-experienced designers and ensure that the design role is consistent across multiple Scrum teams. Lead designers meet with the other project designers on a regular basis (often once a week) to discuss design issues across all teams (see Chapter 8, "Teams," to learn about communities of practice).

Scrum demonstrates—through sprint results—whether the project has enough designers. Scrum teams challenge designers who cannot communicate effectively. A benefit of Scrum is in exposing these problems so that a lead designer will step in to mentor less-experienced designers.

Designer as Product Owner?

Many game development studios using Scrum make the lead designer the product owner for a game. This is often a good fit since the product owner role creates vision, and when we think of visionaries, we often think of successful designers such as Miyamoto, Shafer, Wright, and Meier. Lead designers make excellent product owners for the following reasons:

- Designers represent the player more than any other discipline.
- The product vision is driven primarily by design.
- Design is highly collaborative. Experienced designers should be experienced in communicating vision to all disciplines.

On the other hand, designers often lack experience for some product owner responsibilities:

- **Responsible for the return on investment:** Most designers I've known often need to be reminded of the cost implications of their designs! A product owner needs to carefully evaluate costs against the value for each feature.
- **Project management experience:** Teams accomplish many, but not all, of the duties traditionally assigned to someone in a project manager role. Many requirements or resources that have long lead times require a long-term management view.
- **Avoiding a design bias:** Product owners need to understand the issues and limitations for all disciplines. They cannot assume that everything outside the realm of design "can be handled by others."

For these reasons, it's often beneficial to have a senior producer support the "designer as product owner." A producer can be a voice of reason and cost management.

Summary

Agile reverses the trend of isolation of disciplines. This trend sees designers turning more to long-term plans and documentation to communicate with teams that are ever increasing in size. Scrum practices require the designers to collaborate and communicate face-to-face on small, cross-discipline teams.

In reversing this trend, designers need to embrace the benefit of emergent design. No designer has a crystal ball about any mechanic. The limitations of what is possible prevent this. Instead, they need to ensure that their vision is communicated and open to all potential ideas.

Additional Reading

McGuire, R. 2006. *Paper burns: Game design with agile methodologies.* www.gamasutra.com/view/feature/2742/paper_burns_game_design_with_.php.

Index