The Addison-Wesley Signature Series

A Mike Cohn Signature Book

Agile Game Development

Build, Play, Repeat

Second Edition

Clinton Keith

Praise for *Agile Game Development*

"Clinton Keith has written an excellent book for both practitioners and students. He combines an in-depth analysis of the challenges of large scale game development with hands-on advice on the use of Scrum. His often funny anecdotes illustrate that this guy has really experienced the heat of large computer games projects."

—Bendik Bygstad, Professor in Informatics, University of Oslo

"This book is an *essential* guide for developing creative projects, in an Agile format. There are so many misunderstandings of what 'Agile' truly is, and Clint explains it in a way that anyone can understand. If you are managing creative teams, this is a must-read."

—Brian Graham, VP Product Development

"I had the great fortune to complete my Scrum Master Certification training from Clint just before he published the first edition of *Agile Game Development*. I've still got the copy I bought in 2010, when it was first released. We've stayed in touch and learned and shared a lot over the past 10 years. I love Clint's writing style and hope that the new edition of the book inspires many more, like myself, to continuously learn and grow as Agile practitioners."

—Erik Byron, Game Developer and Consultant

"I wish Clinton Keith could go back and write this book 15 years ago—it would have helped me see things a lot differently. *Agile Game Development* is a one-stop-shop for game teams interested in using Scrum techniques."

—CJ Connoy, Sr. Producer

"Clinton Keith combines his experience as both video game developer and Agile practitioner to apply Scrum philosophy to the unique challenges of video game development. Clint clearly explains the philosophy behind Scrum, going beyond theory and sharing his experiences and stories about its successful application at living, breathing development studios."

—Erik Theisz, Certified Scrum Professional – ScrumMaster

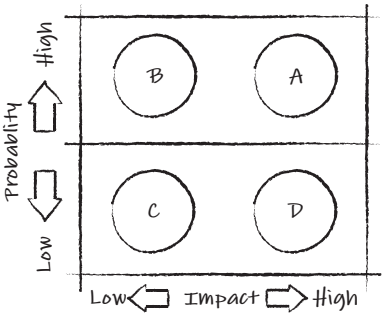

Figure 7.2 *The risk matrix*

A risk matrix is a simple tool. Each area of risk that has been identified is placed on a 2 x 2 map sorted, relative to other risks, by the probability of the risk occurring as well as its impact (cost or schedule) on the game (see Figure 7.2). The risks are then ordered by which quadrant they occupy on the map.

Looking at the matrix in Figure 7.2, the risks in the "A" square are the ones we would be most concerned about. These could be risks like "we can't find enough good programmers to hire and train in time." Risks in the "B" squares are next. Those risks remaining in square "C" can usually be ignored. They are unlikely to manifest, and if they do, their impact will be minimal.

With a set of risks ordered by impact and probability, you can establish a plan to manage them. Chapter 10, "Video Game Project Management," explores an approach for doing that.

Try Running a Premortem

A premortem (Brown S., Macanufo J., 2010) is an imaginary postmortem held at the start of a game. Participants put on their "future hats" and imagine themselves gathering a month after the game has been shipped. They then describe all the problems with the game and with the effort to develop it. These scenarios are used to populate a risk matrix and identify solutions long before these potential problems occur.

The Practice

Small groups of people (five to nine in size) gather to create a poster for the game. On this poster, they write the game's goals (genre, market positioning, major features, and so on) and create the plan (critical dates, cost, team structures, and so on) on one half. The other half of the poster is set aside

to capture "what went wrong." Teams fill the "what went wrong" half with sticky notes, each containing one area where things went wrong with the game in the market or in the effort to develop it.

After a team completes the aforementioned steps (30–40 minutes), one member of the team presents their poster to other teams or stakeholders.

Tips

Here are some tips that can help when running a premortem.

- If there are a lot of "what went wrong" sticky notes on a poster, ask the team to pick the three items that are most impactful to present.
- Identify areas where differences exist in the goals and plan section. This is a great opportunity for a Product Owner to create and discuss a shared vision of the goals and plan for the game.

Evaluating Knowledge

One of the best ways to gain knowledge about something unknown is to perform an experiment. A useful tool for prototypes is using a timeboxed PBI called a **spike**, which limits how much time the team spends working on it before it can be evaluated. An example of this is a two-week prototype to determine whether the physics engine supports destructible geometry. If this spike demonstrates that the value, cost, and risk of implementing the system and toolset is not too great, the Product Owner is more likely to raise the order of a PBI to develop the full feature.

Evaluating Value

Knowing what your players want can be a challenge, especially the further out in time you try to forecast. Although live games can use Key Performance Indicators (KPIs) and frequent releases to guide value over the short term, identifying feature value over the mid- and long term can be nebulous. The following tools can help categorize value.

Kano Analysis

The Kano analysis model prioritizes features into the following market categories:

- **Delighters:** These are features that are rare in other games or have never been seen before. They would be heavily promoted in a marketing campaign and would delight players; for example, massively persistent/destructible MMO environments.

- **Satisfiers:** These are features that players are not surprised by, but enjoy. Competitors may or may not have all of these; for example, collaborative online zombie maps in a first-person shooter.
- **Basic expectations:** These features are not advertised but are expected by the player, and their absence will upset them; for example, save game checkpoints.

Batman Quality!

"Product Backlogs are great for prioritization, but that doesn't necessarily correlate to quality. But we've found a way to measure quality through the Backlog by adding a few more dimensions: customer impact and quality goal. Customer impact would be used to specify how much impact the implementation of the user story would have to the end customer, and quality goal would be used to set several expectations and alignment per quality level.

"You can measure customer impact at three levels:

- **Minimum** is the minimum passable version of the story.
- **Awesome** is making the story good or better than competition or user expectation.
- **Batman** is something noteworthy and amazing, because, well, it's Batman.

"Each has their Definition of Done, which establishes the quality goal. These metrics allow us to look at completed stories and measure the number of stories at certain quality levels, completed by customer impact. If everything is minimal, that gives us a good view of how it may be perceived.

"Having user stories with additional clarity of quality goals allows the ability to do a competitive marketing analysis and make more informed scoping decisions. It is extremely unlikely that any project can implement every user story at the highest level. But it's also not necessary. By having the customer impact measure for items that are low impact, it is most likely safe to implement the minimum version.

"Before adding these measures, all user stories can be seen to be equal in impact and quality. Throughout the project, you can run a Backlog report; using the measures of customer impact and quality on completed features

> will give you the ability to view the current quality state. We can then put a higher weight on the high customer impact items and calculate a quality score for the product. Using this helps make better scoping decisions, to at least give us better odds at delivering the right quality, in the right places."
>
> —Brian Graham, Production, Playful Corp.

MuSCoW Analysis

MuSCoW Analysis prioritizes features into the following categories:

- **Must have:** Features we cannot ship without. If we cut them, we'll fail; for example, online gameplay on a first-person console shooter.
- **Should have:** Features that can be cut, but would impact us if we do; for example, particle effects for a mobile slots game.
- **Could have:** Features we would like to keep, but can cut without much impact to the game; for example, posting game progress on Facebook.
- **Won't have:** Features we will not keep; for example, a small character that follows the player around in the game incessantly giving them advice.

MuSCoW is especially useful for identifying an initial feature set for a live game with a minimum set of features. Its benefit over Kano is to explicitly identify features that won't be included with the stakeholders.

Value/Cost Ordering

The matrix in Figure 7.2 can also be used to order PBIs using cost on the horizontal axis and the value on the vertical axis, as shown in Figure 7.3. PBIs are then ordered through the four quadrants:

1. High Value, Low Cost
2. High Value, High Cost
3. Low Value, Low Cost
4. Low Value, High Cost (don't bother with these!)

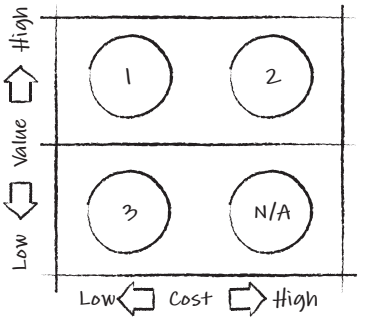

Figure 7.3 *The value-cost matrix*

Defining "Done"

Each Sprint, teams commit to completing a number of PBIs from the Product Backlog and demonstrating that these are done in the Sprint Review. However, defining what *done* means can be challenging. There are many bad examples of what "done" means in the game industry, such as

- "It runs on my PC."
- "It looks good in Maya."
- "It compiles."
- "I checked in the asset."

These loose definitions result in debt piling up in the game. Let's look at the types of debt that exist in video games, how debt is managed, and how Definitions of Done can help.

Types of Debt

Ward Cunningham, one of the authors of the Agile Manifesto, once likened the problems with technology, such as bugs and unmaintainable code, with financial debt—it grows the cost of payback over time.[2]

2. http://wiki.c2.com/?WardExplainsDebtMetaphor

Debt results in an ever-expanding amount of work that needs to be addressed before the game is deployed. Numerous forms of debt exist for game development, such as

- **Technical debt:** Bugs; slow and unmaintainable code
- **Art debt:** Assets such as models, textures, and audio that need to be reworked or improved
- **Design debt:** Unproven mechanics waiting to be fully integrated into the game to prove their value
- **Production debt:** The amount of content that the game needs to be commercially viable (for example, 12 to 20 hours of gameplay)
- **Optimization debt:** Work waiting to be done to prove the game can run at acceptable frame rates on the target platforms

If unmanaged, the amount of work to address debt is unpredictably impactful and results in crunch and a compromise in quality as key features or assets are dropped in favor of maintaining a schedule.

Game Development Debt Example

One typical example of game development debt is in the creation of characters. Characters are often designed, modeled, rigged, and animated wholesale before we really know what we want them to do or before we know their budget. This is done because of schedule pressure or a plan that optimizes discipline allocation over the delivery of working assets.

As a result, by the time we figure out what we want the characters to do, we realize that the requirements for rigging and animation have changed, and instead of having to re-rig and reanimate one character, we have to do it for 20.

Demonstrating one character behaving the way we want in the game before mass-producing the remaining characters would have avoided this debt. If the character production schedule is a critical path for the project, then prioritize it as a risk and order the work that removes the uncertainty of character budgets and requirements early.

Managing Debt

The benefit of managing debt is to reduce the cost of development, avoid crunch, and avoid compromising quality. This is easier said than done due to the pressure of delivering quantity over quality. When we estimate the time to deliver scope, we don't