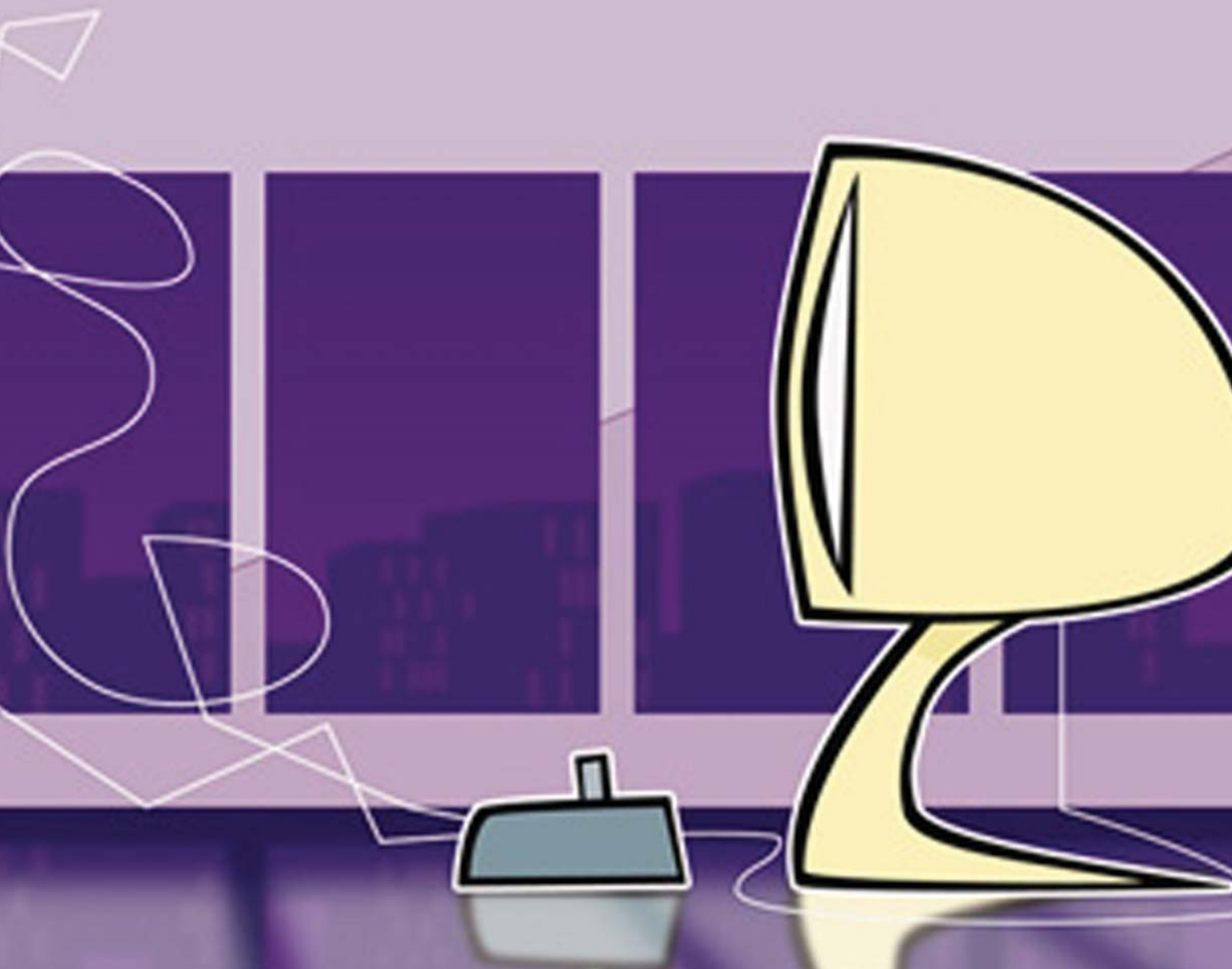




Get in the Game

By Clinton Keith

*What others
can learn
from game
developers*



More than twelve years ago I left a defense industry software engineer job to become a video game programmer. The higher level of technology and quicker pace of product development in games—about ten times greater, in terms of lines of code—shocked me. The game industry attracted some of the best, most highly motivated programmers, and the latest software engineering metaphors were discussed with a passion. Yet for all the enthusiasm, there was a nearly complete lack of—and a high level of disdain for—process in project management.

In just three decades, video games have gone from a niche toy industry making Pong boxes to a branch of the entertainment industry larger than Hollywood's box office. The change

has increased development teams and budgets exponentially.

Both of these factors have led to massive industry problems. One is the sometimes egregious abuse of people who work in games; crunch times lasting a year are not uncommon and have resulted in lawsuits for overtime pay. Project delays and cost overruns are also common.

In response to these challenges, we at High Moon Studios have taken lessons from other industries and adopted agile methodologies to stay competitive and deliver on time and within budget. We've had to make changes to account for unique development environment issues, but they haven't broken the vision of what it means to be agile. We focus on

communication and iterating on an emerging product from the start, and we can react to changes from many different directions to make the product better.

Background of Video Game Development

When I entered the video game industry, it was less than twenty years old. In the late 1970s and early 1980s, it had experienced significant mainstream growth. Coin-op arcades—and later consoles that re-created the experience at home—were raking in huge amounts of money.

During this time, a team of one to three people could create a game in about three months. The hardware consisted of 8-bit processors running at a few hundred kilohertz, 16 K of memory, and low-resolu-

For a video game the value is fun ... Fun is very difficult to define in a document.

tion (256 color) 2-D graphics. Individual heroics were common because the consumer demanded that the hardware be pushed to its limits, though the resources were limited. Often the game art was created by the programmers! The main obstacle for development was that mass producing arcade cabinets—or cartridges for home consoles—was far more expensive than creating the games. As a result, development was iterative and incremental. Each game had to show value improvement at frequent intervals or risk cancellation. For each game released, a dozen games might be cancelled, yet the rewards for making a hit game were substantial.

The high profits and exploding demand created problems, namely a relaxing of quality control to allow more game releases. The result in the mid-1980s was the first and to date largest crash for the game industry. Companies like Atari lost hundreds of millions of dollars, and landfills received tens of thousands of cartridges that no one wanted to buy.

Innovation in home consoles helped the market recover quickly, and sales continued to grow with more control by the console hardware manufacturers. Nintendo was a major catalyst in the industry's recovery, introducing its hugely successful home console and ensuring stringent quality controls with a seal of quality on all software released for the system.

The major pressure for game development soon became the growth of hardware power. Within a decade after the crash, consoles had processors running at more than 20 MHz, 4 MB of RAM, and NTSC-resolution 3-D graphics capability. Skilled artists were hired to create highly realistic art, and teams of programmers took on specialized areas of technology such as graphics, audio, and artificial intelligence. Games required a year of development with teams of

more than a dozen people.

At this point the informal incremental and iterative development process began to break down for two major reasons. First, communication among a dozen specialists did not flow as easily as among three programmers. Details of development could easily slip through the cracks, resulting in increased waste. Second, the cost of development began to outstrip other costs, such as manufacturing and marketing. Companies that provided the funds couldn't cancel a dozen games for every game delivered and still make money.

The reaction was to introduce more structured, traditional methodologies to game development, the most common being waterfall. Waterfall was a popular method used in other industries that had experienced large growth in team size and project development cycles. To manage that growth, the method divides projects into a number of phases—roughly: planning, prototyping, development, integration, and testing. In the initial phases, the product and its attributes are defined through documentation, which more than a dozen developers will follow to produce the final product.

Waterfall sacrifices the ability to iterate. That sacrifice was an acute problem, as it is difficult to know the value of a game and its attributes at the start of a project. For a video game the value is fun, and many of the elements that help define that value only become evident in demonstrable iterations of the game. Fun is very difficult to define in a document.

Moving to Agile

So given these problems with waterfall development, there was a strong desire to move back to an iterative and incremental methodology at High Moon Studios. The major barrier to this was team size. We needed a more formal methodology—one with a set of practices that would

restore the team atmosphere we had had when team size was ten people or fewer.

We found Scrum and, within a month of adopting its practices, saw significant improvements in team productivity and morale. The team soon recognized another improvement: The game we were developing began to take shape, with the biggest revelation being how playable and fun it had become in a short time. Later, our programming team adopted Extreme Programming (XP) practices and saw additional significant benefits.

Challenges

When we adopted agile, we set a goal to use all of its practices. Our caveat was that we would modify a practice only when we understood its principles. In fact, it took quite some time before senior management understood the principles of agile well enough to stop making wrong decisions—many of which would have been correct for a company practicing waterfall but were incorrect for agile.

After months of working to apply agile and understand the reasons behind its practices, we began to adjust to meet the challenges we encountered.

Hard Deadlines

Video game development is largely driven by hard deadlines. Most deadlines are seasonal and driven by marketing. Marketing deadlines can be most critical to a game's success. Many mainstream games rely on a holiday release, the largest season of the year for game purchases. For games based on film licenses, it is crucial to set a deadline that allows the simultaneous release of the game and the movie.

With a fixed schedule, the only truly flexible element left for development is the scope of the game. Budgets may be flexible, but throwing more people at a late project usually makes things worse.

Controlling scope is where the benefit of an iterative “value first” approach to making games becomes apparent. Using agile, game development proceeds in an iterative manner that operates by priority, with the most important features being addressed with every successive iteration. When the team is running short on time, it can stop adding features with confidence that those not implemented are less important than the ones already in the game.

When using a waterfall method for development, feature implementation isn't necessarily done in order of value. For example, the technical team may want to focus on a complex animation system before an artificial intelligence (AI) system. If the animation takes longer than expected, then it will use some of the scheduled AI development time, so the AI system may not be developed into something challenging to the player. Additionally, the animation system might have functionality that won't be used by the simpler AI system. In the iterative approach, the objective would be to implement just enough animation and AI to achieve the most important features of the game before adding additional features. In other words, the goal is to develop functionality as needed, not according to a preconceived plan.

Contracts

Video games are created by developers, who sign contracts with publishers. The publisher's job is to pay for game development and then market, mass produce, and distribute the game.

With waterfall, many publishers want to see an up-front, well-fleshed-out plan of how a game will be developed and a timetable of when each function will be delivered. These deliveries, or “milestones,” are associated with payments that fund the game studio. Milestone payments are a developer's lifeblood, as receiving them is critical to survival. Essentially contracts, the milestone schedules create the incentive for developers to ensure that they implement every feature defined, regardless of its value to the game. But the milestone schedules also create obstacles in the relationship between publisher and developer when changes are requested.

One challenge in adopting agile has been creating contracts that allow for the method's practices. Agile planning is flexible and only defined in the near term, usually three months. While this short-term scheduling once horrified many publishers, they've begun to see the futility of their previously required level of up-front detail. Trust between developer and publisher is another hurdle. It needs to be established over time, as the publisher recognizes the developer's productive pace and commitment to quality, and the developer begins to believe in the publisher's vision and integrity. Trust strengthens the relationship and allows a better game to emerge.

Trust does not mean unwritten contracts, but that they need to reflect the iterative, flexible approach of using agile. For example, it's difficult to define when the developer has breached the contract if the game isn't fully defined before development starts. One approach addressing this is to create a number of smaller contracts, each covering a three- to six-month period known as a “release.” A release is a version of the game that could potentially be shipped to market. These short-term contracts require good faith between developer and publisher, but if the game is showing increasing value or fun over each release, it will be easy for the publisher to recognize the developer's progress.

Specialized Roles

Another challenge to creating truly agile teams is the existence of specialist roles. Traditionally, agile teams might have specialized programmers who, at the very least, understand one another's job and can help each other to a certain degree. With game development teams, artists, game designers, and programmers have entirely separate roles. An artist looking at a programmer's work, or vice versa, sees what can be described as a “black art.” This requires a lot of faith and trust among teammates.

The heterogeneous mix of a game development team makes some of the basic agile practices more challenging. Self-organization is more difficult because artists may not choose team programmers using the same criteria that program-

mers would use. The same can be said for the benefits of mentoring and sharing tasks, which are more difficult in a cross-discipline team environment.

Ideal team size is more difficult to achieve, as well. With agile there are approximately eight to twelve people on a team. Beyond this limit, the lines of communication get too complicated for the team to self-organize. With several disciplines on a team, it's more difficult to get critical mass within each discipline. For example, four team programmers is the minimum, with two pairs in an XP environment. Six is ideal, but if each discipline has six people, then the team becomes too large. The solution is teams that are skewed slightly toward one discipline over the others. This can create challenges too, but it keeps the teams smaller.

We first adopted Scrum only within our programming department. The artists and designers were customers of the Scrum programming teams' creations. We quickly learned that hundreds of progress impediments were occurring during every sprint due to lack of customer input. When we realized that Scrum teams needed to be self-contained to solve most of their impediments internally and immediately, our artists and designers joined the teams.

Supporting the Disciplines

Before agile, our project teams were split mostly into teams of programmers, designers, and artists. This was great for shared disciplinary skills, but it increased the overhead required to get things working in the game. When we transitioned to agile, this became transparent and needed to change. There was initial resistance, which affected our eventual transition to feature teams.

We first formed “functional teams” dominated by one discipline. (One example is an AI team composed mostly of programmers.) Problems emerged after a few months. The AI team was creating its work in a vacuum. The AI requirements weren't being iterated and explored in a real game but rather developed by a programmer-dominated team that focused on delivering an architecture

that all AI-dependent teams would need. After each iteration we observed architectural progress but not an increase in real game value.

We next moved from functional teams to “feature teams.” Feature teams focus on a vertical slice of game play, such as a shooting or driving mechanic, and require an interdisciplinary group of developers. The goal is a self-contained team that can address every part of development required to iterate on its vertical slice.

With our move to feature teams, the development of core technology areas became highly distributed. For example, we shifted from one functional team’s creating all AI for the project to one or two programmers creating AI on each feature team. This created problems as well. First and foremost was that technology development fractured. While fine for early exploration, this was problematic down the line. We were behaving as though we were creating separate games. We knew we would eventually need one AI system for the entire game. Also, we saw that feature teams would compete with one another for resources. This did not create a sense that all teams were part of one project.

The solution was to form virtual teams around each discipline. The programmers formed a team with a ScrumMaster that met at least once a week to discuss the direction of the team’s technical effort. The programming ScrumMaster met with the ScrumMasters of the other disciplines (art and design) to form a vision for game development. This helped the separate teams better communicate and understand that they were all a part of the final product.

Development Phases

Agile game development hasn’t avoided having some project phases such as those outlined by waterfall. For instance, the concept-development phase is needed to explore the creative possibilities and limitations of the property so that customers can visualize overall creative goals. A game project cannot fully explore itself during development. A customer will have marketing criteria or requirements that go with a license

for a game title. This requires pre-visualization and concept art.

The production phase occurs during the last half of the project, when mass production of certain assets—usually characters and levels—takes place. Before production, the game play can be explored with fewer constraints. We create a small number of representative levels and characters. Once we discover the elements of fun that will govern the creation of these production assets, we lock them down (see the StickyNotes for an example).

Crunch Time

The game industry is infamous for working people overtime. Recently, there have been high-profile lawsuits filed on behalf of game development employees, sometimes by families impacted by their loved ones’ absences. Fortunately, this hasn’t been a problem for us. We essentially eliminated extended crunch time by using the empirical data as part of the Scrum method practices. In Scrum, the team estimates on a daily basis its remaining work for the iteration being developed during a sprint. The daily change in the number of iteration hours remaining is called the “velocity” and is plotted on a chart. Many factors impacting velocity, such as technical impediments, vacations, seating arrangements, etc., can be seen on this chart. It also shows the effectiveness of the team.

Even with Scrum, we’ve had occasions when a team was forced into crunch time for an upcoming sprint deadline. This occurred because management bypassed the Scrum rules and, rather than having the team commit to work, insisted that it commit to a schedule of ten hours a day, six days a week. At first the velocity slope increased, but over several weeks we saw the velocity fall until it was below the pre-crunch-time level. This empirical measure showed us that factors other than time contributed to product velocity. Overworked people become tired and make more mistakes, which manifest themselves as game bugs or art mistakes that require extra time to correct, resulting in reduced velocity.

This was an important lesson for us.

Teams may decide to work extra time during an iteration if, for instance, they are faced with not achieving a goal. But management no longer dictates it.

Summary

We continue to explore how agile applies to video game development. The stakes are high. Improved methodology can reduce development cost, resulting in reasonably priced games. It also helps us avoid the increasing risk-averse trend, where fewer companies are willing to take on creative challenges. Games based on movie or sports licenses and sequels based on previously successful properties currently make up the bulk of game products on the market. Titles that introduce new intellectual properties or novel ideas are seen as too risky for the sizable investment in development—a dangerous path for our industry to pursue. Entertainment consumer dollars and audience attention are at a premium. A trend that decreases the overall value and enjoyment of video games will hurt everyone in the industry by shrinking our market. Agile provides the framework to explore new experiences and iterate on novel ideas to “find the fun” in games early and often and create better ways to entertain and grow our audience. **{end}**

Clinton Keith has gone from programming avionics and other software for advanced fighter jets to overseeing programming for video games. He is currently the chief technology officer at High Moon Studios, a video game developer in Carlsbad, CA. In his video game career Clinton has developed or directed development on titles such as Midtown Madness, Midnight Club, Smuggler’s Run, and Darkwatch to name a few. Prior to developing games, Clinton served as software engineer on undersea technology at Raytheon and on the YF-23 and YF-22 avionics for TRW.

Sticky Notes

For more on the following topics, go to www.StickyMinds.com/bettersoftware

- Pre-production elements of fun