

# Why Use Scrum?

By Clinton Keith

---

## *The Problem*

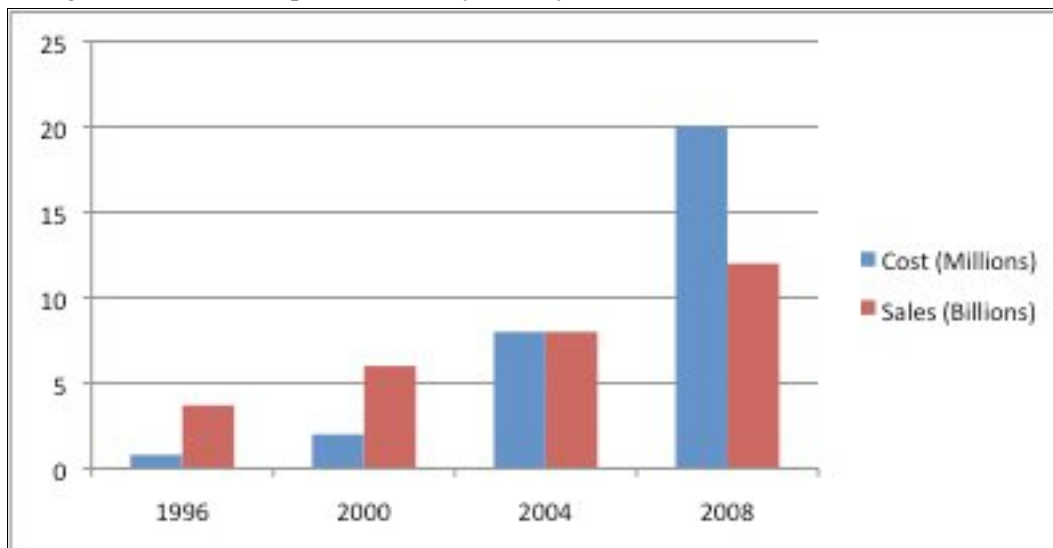
If I owned a game development studio right now, I might not be able to sleep at night.

I'd lay awake thinking about how much money it cost to run my studio per day. My typical project has over 60 people working on it at any one time. That's about \$50,000 per day per project. That's a lot of money. If that project takes two years it will cost well over \$10 million dollars. Add the publisher's cost of marketing and distributing the game and that project would have to sell over a million units to break even. How many games sell over a million units?! Not many.

Game development has never been the model of efficiency. It requires very talented people with creative ideas. It's not the place to set up a factory where you hope to roll out hit after hit. A great game emerges like magic. Creating a great game is not a predictable process.

Since the beginning of the video game industry, we've been driven by hits. We waste a lot of money making bad games and make it all back (& lots more) when we manage to make a hit game.

These days the stakes are different for each game. The "hit or miss" model no longer works because the numbers don't work the same. A decade ago, the million-dollar budget was rare, because the teams were small and technology was far more limited. Take a look at Figure 1. It shows that although the video game market has grown steadily the past decade, the cost of developing games has skyrocketed at a far higher rate. Ten years ago, a single hit could finance ten failures. We are rapidly approaching the day when each game must make a profit. We may already be there.



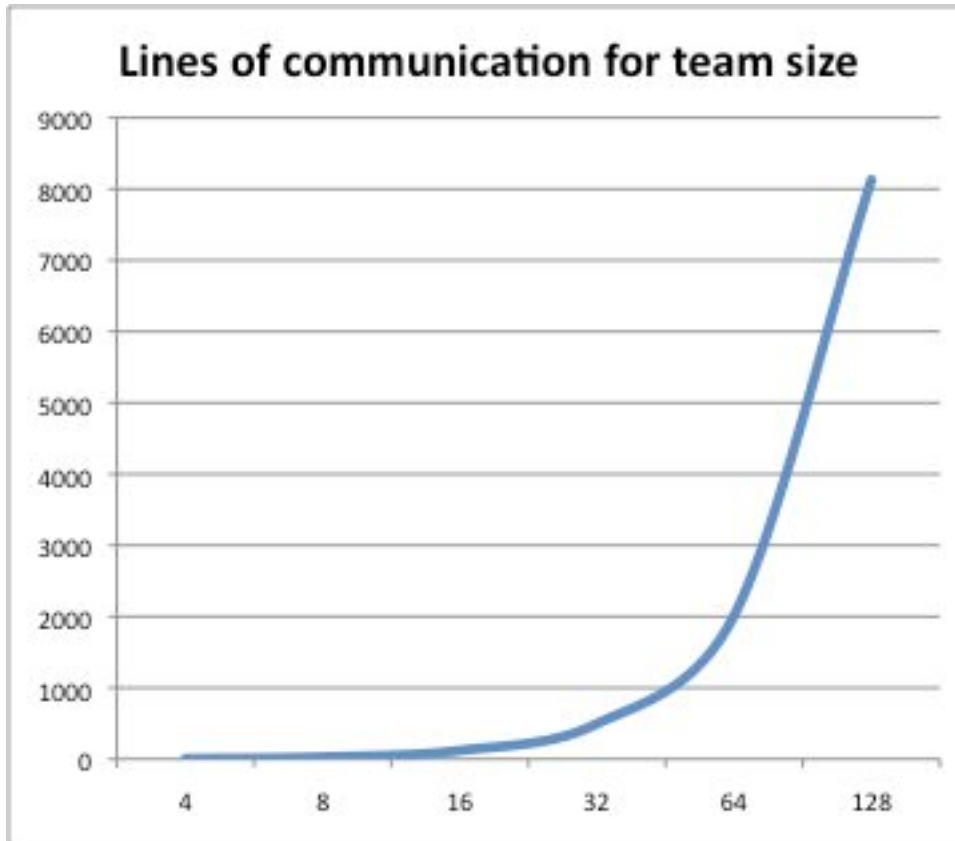
But let's back to that \$50,000 dollars a day. That's what is keeping me awake. Did we add \$50,000 worth of game today? Will we tomorrow? Today wasn't such a great day. The build has been very unreliable for a week now...ever since we got that latest version of the engine. We're bleeding money.

The last ten years has seen a number of improvements in how we make games. The tools for making games such as Maya and Visual Studios are a magnitude better than previous versions that run under Windows 95. With all the progress it still seems as though we're losing ground. This has to do with the ever increasing demands of the hardware and the market. While 5000 polygon scenes and 2 megs of texture memory were the norm in the mid nineties, we're two orders of magnitude beyond that today. Today's market demands a high level of detail in games.

This details has driven up the effort of making games. Today, teams of over 100 are common where a decade ago, a team of 20 was large. It's clear that the cost of so many people on a project is much larger, but unfortunately we have come to learn what other industries have known: Teams don't scale in effectiveness as they scale in size. A team of 100 people aren't five times as effective as a team of 20.

Why is that? Part of the answer is that it's human nature. We've spent hundreds of thousands of years organized in small families of about 10 people and larger tribes of about 30-50. Our brains are wired for working differently for each group size. You can see this reflected in one of the oldest human organizations, the military, which has learned the lessons of how people work best in various sized groups. The military organizes at different size groups based on how well people work within those groups. For example, the "Squad" consists of about 8-16 soldiers that can communicate at the highest levels of efficiency. The next higher group size is the "Platoon" which consists of about 30 soldiers. You can still know all the other people in a Platoon and be able to communicate directly with them, but not as effectively as on a Squad. Above this is the "Company" of about 200. At the Company level we find that we need more of a hierarchical organization to start handling the communication that needs to happen between various members of the Company. Each higher level comes with a reduction of communication efficiency because of the overhead involved.

The other part of the answer is the complexity of communication that needs to occur between people on the same team. Team communication complexity is referred to as an "N-Squared Problem"  $((n^2-n)/2)$ . This means that the complexity of communication goes up as square of the team size. For example, a team of 20 people has a communication complexity (190 lines of communication), which is four times greater than a team of 10 (45 lines of communication). This is illustrated in Figure 2.



The other part of the problem with creating large projects that require large teams is the attempt to create huge plans that are meant to manage the risks of the project. We create huge plans to try to predict how we will address the complexity of large projects. Unfortunately these plans give us a false sense of security when we rely on them too much.

When General Dwight Eisenhower was asked about the plan for D-Day, he was reported to say that “The Plan is useless, but planning is essential”. Eisenhower knew that over one hundred thousand soldiers, thousands of missions and hundreds of individual battles were never going to follow any plan exactly, even from the start. However the knowledge gained from the extensive planning exercises leading up to the invasion were absolutely necessary to allow the plan to evolve as the reality of the war emerged.

While game development has not yet reached the scale of the Normandy invasion (it’s projected that it will by the time the Playstation 6 is out), our plans for developing current generation games has reached the same level of fragility. Plans go wrong from the start, but being prepared with knowledge is absolutely essential.

One particular problem we have with game development is the fuzzy definition of our major project aim: to create “fun”. I’ve never seen a 300 page plan for a game that clearly communicated to me how the game was going to be fun.

The problem we see with big plans for games is that we we inadequately handle the problems associated with reality departing from the plan. Large plans take a large investment in time, but we never properly invest the time needed to maintain those plans as the game emerges. When the plan sufficiently diverges from the upfront plan we are essentially flying blind. We need to find a way to spend less time on the plan and more time on planning.

---

## *A Solution*

What game development needs is a new way to work. Unworkable plans and team sizes have to give way. However we can't go back to the days when four people could make a game without a page of planning. Fortunately we're not the first industry to have encountered this problem. Large software development projects that require hundreds of people have existed for over 40 years. We can borrow ideas from those in how to deal with large teams and uncertainty.

Originally, the complexity of large projects was addressed by increasing the level of planning and adopting phases of development. This is often referred to as the "Waterfall Methodology". Waterfall turned out to be a disaster. Trying to plan away all uncertainty gave your project a false sense of security and hid problems until late in the project when they most expensive to fix. As a result, more projects adopted a more iterative and incremental approach to development. The idea of iterative development was to build increasingly better versions of the product that included a bit of planning, a bit of coding and a bit of testing. This way problems were found early and assumptions about solutions were proven or disproved.

Many game development projects use a blend of Waterfall and Incremental and Iterative approaches. A detailed project plan and schedule is adopted, but iterations or milestone builds of the game are released the publisher to prove progress and address risk earlier.

Agile methodology takes the incremental and iterative approach a step further. Agile places greater value on the product than the product plan and on communication between members of the teams and between the team and the customer in addition to the contract. Many assume that agile eschews all planning in favor of just iterative development, but this is not true. Agile places the emphasis on planning, not the plan.

Agile itself does not have any practices, but merely defines a set of values and principles all agile methodologies that call themselves agile. These can be found at [www.agilemanifesto.org](http://www.agilemanifesto.org).

## **Scrum**

Scrum is such a lightweight agile methodology that it can't easily be called a methodology. It's often referred to as a "framework" of principles and practices that wrap how you work. Scrum addresses the major problems of large teams working on complex projects through simple practices.

In a nutshell, Scrum consists of small cross-discipline teams of about 7 to 11 people that add a vertical slice of features to the game every 2-4 week iteration.

The team organizes themselves and decides how much they can accomplish every iteration. They break down the work into tasks that they estimate. They meet every day for 15 minutes to discuss the progress of the iteration and any problems they are encountering among themselves.

The core philosophy of Scrum is that small teams of people produce the best possible results. This is based on the fact that small groups of approximately eight people can communicate at the highest level of efficiency.

Unfortunately a team of eight people cannot make a major console or PC game title these days, at least in any conceivable amount of time. Therefore a typical game project using Scrum will have many Scrum teams working in parallel.

A key principle of Scrum is that the order that work done for a project is based on a prioritized list of features. This list is called the product backlog and the priority is mainly based on the value of each feature to the buyer of the game.

This is made possible by the idea that each iteration of development, called a “Sprint” in Scrum, brings a level of completion. Every iteration includes the elements of a complete project (design, coding, asset creation, debugging, optimization and tuning).

The results of each iteration are used to refine the priorities of the features in the product backlog. For example, a team could produce a jump mechanic for the game in an iteration and find that the mechanic adds so much value to the game that further enhancements to the jump, that would have never been pre-planned, could be added to the next iteration.

Tasks in Scrum differ from those on typical projects. In Scrum, the team will break down their tasks from features on the product backlog themselves rather than having a lead or manager assign individual tasks and estimates to them. The main reason for this is that the team will take ownership of their work to be completed in an iteration (Sprint).

Giving the teams a level of ownership of their goals and how they operate as a team is a key principle. This has to do with how people behave as owners. Take, for example, how people treat their own car versus a rental car. With ownership, teams take their commitment to their goals far more seriously than they do when work is directly assigned to them.

Teams that make commitments as a team accept that they will succeed or fail as a team. As a result, you begin to see peer pressure emerge as a power force on Scrum teams. Individuals that might have been unreliable in meeting goals will find a new passion in insuring that their work is not going to be the cause of their team failing. Returning to the military analogy used earlier, soldiers will more likely perform heroic deeds to protect their fellow soldiers in the trench next to them than doing it for their country. While we don’t expect game developers to jump on live grenades, we’ve seen the “all for one and one for all” attitude have far more weight than managerial reviews.

One of the main confusions about Scrum is about planning. Many people have the false impression that agile methodologies like Scrum eschew any planning and that agile teams run a risk of endlessly iterating and never finishing. These fears aren’t unfounded. Some teams have seen that traditional project plans have limited value and are attracted to the idea of avoiding planning when they start using agile. Many of these teams do end up iterating in random directions and not getting anywhere.

True agile planning follows Eisenhower’s famous attitude about planning. Planning is an essential part of agile, but agile projects do not rely on a fixed plan. In fact, a properly run agile project spends more time in planning over the course of the entire project than a Waterfall run project does.

On an agile project, plans are considered snapshots of what we know. Plans change to reflect reality rather than the other way around. Additionally we do not try to plan away uncertainty on an agile project. The goal is to very clearly identify what we don’t know, which is where the risk lies on an agile project, and use the uncertainty we identify to prioritize the order of the work.

A key strength of Scrum is that its practices create visibility and during the game’s development. The main area of visibility is in the progress of the game every iteration. Since the team will work on a vertical slice of features every iteration, it should be able to demonstrate an improvement in value of the game at the end of every iteration as well. The bottom line is that the game should be more fun every

time. If it's not, then the team and customers of the game should change direction a bit. If they find something fun, they should "double down" on what they found.

Another area of visibility is created on a day-to-day basis with the team. The team meets daily for 15 minutes to discuss their progress and problems. This meeting is key because it creates visibility for progress and visibility for problems.

Visibility of progress is created by a task burndown chart that shows the team their day-to-day progress towards the completion of the goals they committed to. This chart can be used to identify problems early on in the iteration and also identify when changes to the way the team works produces increases in productivity or not. For example, when a team that sits apart in their studio co-locates, they will typically see approximate 20% improvement in productivity due to this one change.

Visibility of problems is probably the most important aspect of Scrum. Fred Brooks, the author of the book "Mythical Man Month" was once asked "How does a large software project get to be one year late?" His answer was "One day at a time! Incremental slippages on many fronts eventually accumulate to produce a large overall delay". The daily Scrum meeting asks all attendees to report problems that affect their progress. By focusing on solving these problems as quickly as possible, Scrum addresses the biggest cause of delays.

So how would agile help the CEO of a game development studio? The first thing to point out is that Scrum is not going to convert your studio into a hit factory. Great games are made by talented developers. If you don't have the right people, Scrum isn't going to do you much good.

Scrum will make development at your studio a lot more transparent. When you adopt Scrum, you will experience a large surge of problems reported on a daily basis. This is due to the practices which encourage transparency. Some executives can't handle this transparency very well. The flood of reported problems is interpreted to be a signal that Scrum has broken their studio and they need to revert back to their old methods. Others will tinker with the practices a bit until the noise quiets down. An example of this would be eliminating the daily scrum where all problems should be reported. Studios that are successful with Scrum understand that the transparency is extremely valuable and work to support it.

Another value to the studio CEO is the constant iterations of vertical slices of the game that show the value of the emerging game. There is no excuse that you should wait until alpha to know whether your game has a chance of being any good or not. The vertical slices being done should be prioritized not only on the value to the game, but to address concerns of risk and cost to the overall project. Are you concerned about the game being able to run on the PS3? Prioritize PS3 work early!

The bottom line is that Scrum creates transparency, which allows you to make common sense decisions. If you ignore or do nothing about this transparency, then Scrum will do you no good. Scrum should never reach a state where no problems are reported either. It opens the door for continuous improvement that should become part of your culture. As you've seen in the last decade or more, the pace of change is not fixed. The way you make games should never be fixed either.

Clinton Keith is an independent agile coach Certified Scrum Trainer with 15 years of video game development experience. Clinton introduced the game industry to Scrum in 2003 and Lean/Kanban in 2006. He has coached teams at many studios. He is the author of "Agile Game Development with Scrum" which will be published in early 2010. His website is [www.ClintonKeith.com](http://www.ClintonKeith.com).