Beyond Scrum: Lean and Kanban for Game Developers

By Clinton Keith

[If you've discovered the value of Scrum agile development while making your game, expert Clinton Keith outlines Lean and Kanban, two ways you can be agile during all phases of the game development process.]

Many games teams that adopt Scrum quickly discover its value. The improved velocity of introducing value, or fun, to the game at a regular iterative pace shows where the project is heading and allows the team and customers to react by improving the game frequently. However teams find that when they enter production, the value of Scrum diminishes.

Many teams abandon some Scrum practices late in a project's life cycle and return to more traditional waterfall practices. They call this approach "a blend of Scrum and Waterfall". This article explains the reasoning behind this and introduces the concepts of Lean Production and Kanban as an alternative to adopting waterfall practices.

Lean and Kanban can answer the issues with Scrum that many teams face, but they don't require the team to abandon agile methods. These practices are based on real world production experiences which showed a 56% improvement in the cost of level production.

In most agile projects outside the game industry, there are no phases of development. There are no concept phases, pre-production phases or production phases. These projects start with releases, and every release delivers a version of the product to customers. Think of applications like Firefox that release a new version every month or so. Most games have a single release that requires years to achieve.

Eliminating phases is a big benefit of agile; waterfall phases such as the testing phase force the critical activity of testing to be postponed to the end of the project, where fixing bugs is the most costly. Planning phases at the start of projects attempt to create detailed knowledge about what features will be fun and the work associated in creating them. Unfortunately the best knowledge comes from execution, which is why highly detailed pre-planning fails.

For many games however, there is still a need to have phases within the game. There are two major reasons for this:

There is a minimum bar to the content being delivered regardless of the quality. Sixty-dollar games must deliver eight to 12 hours of gameplay. This represents the major portion of the cost of development and occurs after the gameplay mechanic is discovered. This requires a pre-production phase to discover fun and a production phase to mass-produce the assets for the eight to 12 hour experience.

Publishers have a portfolio-driven market model. This constrains the goals of the games that they fund. In order to gain publisher approval (which includes marketing and often franchise/IP owner approval), developers need to create a detailed concept treatment during a concept phase at the start of a project. Developers are then unable to stray too far from this vision throughout the project.

Pre-production allows more freedom to iterate on ideas and explore possibilities. During production, we are creating thousands of assets that depend on what we have discovered during pre-production. These assets create a cost barrier to change during production.

For example, consider a team in production on a platformer genre game. Platformer games (such as Nintendo's Mario series) challenge the player to develop skills to navigate treacherous environments. The production team will create hundreds of assets that depend on character movement metrics such as "how high the character can jump" or "the minimum height that the player can crawl under". Production assets depend on these metrics.

If these metrics are changed in the midst of production, it can wreak havoc. For example, if a designer changes the jump height of the character, hundreds of ledges or barriers would have to be changed. This can create a great deal of wasted effort during the most expensive phase of development.
It’s critical to discover and lock those metrics during pre-production. This doesn’t mean that we can’t be agile during production. How we are agile does change. Instead of using an iterative and incremental process such as scrum, a more incremental process such as Lean is more applicable.

**Sprints and Production**

Asset creation is deterministic and sequential work that does not fit the Sprint iteration cycle very well. If we think about production as a factory assembly line, then the two to four week iteration cycle doesn’t make as much sense. Factories don’t empty the assembly line every four weeks and determine what to build next.

Assembly lines have things rolling off much more frequently and require incremental improvements instantaneously. The rate that completed assets roll off the line becomes the new heartbeat of the production team.

---

**Scrum Task Boards and the Production Team**

At the start of a Sprint, the Scrum team will commit to completing a set of tasks that they estimate. Those tasks are placed on a task board that everyone reviews every day. Many of the tasks can be worked on out of sequence, or in parallel.

If one task is held up waiting for another task, then work can continue on other tasks. This organic flow of task execution fosters communication among the team and prevents impediments from stopping them cold. Scrum task boards are great for showing a large number of tasks that can be executed out of order for a number of Sprint goals.

However, when we have a long string of sequential tasks that must be completed in order, then we lose some of the benefits of parallel execution of tasks. Tasks have to be finished in order and work must flow through in a predictable way to ensure that the many specialists we have on a production team are not starved for work. Scrum task boards fail to represent this flow of work through long assembly lines of production.

**Scrum task boards represent three or four states of a task:**

- Not started yet
- In progress
- Needs approval
- Done

This is sufficient for many tasks in preproduction. There is a lot of back-and-forth between different disciplines behind the scenes that occurs while a task is in progress, and that’s fine. However, when we enter production, we can have a long chain of tasks that need to occur before we see some production assets in the game. Take, for example, the steps that need to occur for a single level to appear in the game.

This is a simplification of the large number of tasks and hand-offs that need to occur for every level in the game. Each task in this stream has to occur before it is handed off to the next step.

If one step in the stream fails or is delayed, the repercussions will travel through the rest of the stream. Let’s apply this to a Scrum task-board as a set of tasks:
This task board above tells us that the script is done and concept art needs to be approved before level design can be started. One problem immediately appears -- the visibility of the entire stream cross-dependency isn't shown task board. Perhaps the concept art approval is stalled.

What does this mean to the other tasks? It means that they are all delayed! Who pays for this delay? The person doing the tuning pass, perhaps even the audio design. The main benefit of task boards is to provide visibility to the team about the work they are doing. In this case this visibility is lost due the inherent dependencies in the stream. They just aren't visible here.

Another problem with representing streams on the Scrum task board is: what are the disciplines at the bottom doing while the first set of tasks are being worked on? What are the audio designers doing while the concept art is awaiting approval?

They could create environmental sounds or some other filler work, but that isn't the most efficient use of their time. Scrum task boards empty at the end of every Sprint. In production, we don't want to empty the production line. We want it to be continuously filled so that everyone in the stream has work to do every day.

**So What Use is Agile During Production?**

We need to remove some of the iterative nature of development and become more prescriptive during production. We don't want to abandon the ability to react to change, however. Production is never 100% efficient.

We are never able to predict every potential problem. We can find massive improvements in our pipelines right until we ship the game. For this reason we don't want to abandon agile entirely.

If we establish fixed schedules and deadlines, the best we can hope for is to meet those schedules. Unplanned problems will continue to appear and threaten those schedules. What we need are practices that are still agile; practices that anticipate not only change but focus our attention on continually improving how we produce assets.

**Lean Production and How it Applies**

Lean Development and Production has roots going back to Toyota in the 1940s. During the 1990s, many car manufacturers and other manufacturing industries adopted the principles of lean thinking.

This past decade has seen its adoption in many industries that aren't considered in the traditional manufacturing arena, including software development. Lean principles concentrate on eliminating waste, delivering fast, empowering the team.
and seeing the whole (Among others benefits -- see the book references at the end of this article).

We can apply these principles to game development as well. Like automobile manufacturing, in production we have long chains, or streams, of work that need to be performed by specialists in order.

Like automobile manufacturing, the cost of labor and mistakes are by far the greatest costs. We need to employ the full skills of everyone "on the line" to improve what we do and how we do it. The automobile industry discovered decades ago that assigning everyone on the line a fixed set of tasks does not achieve the best results.

---

**Leveling Production**

Leveling production is a Lean technique for reducing waste and smoothing out the fluctuations of production. This allows us to create production assets a constant a predictable rate.

Much of the time we spend in production is wasted on work or activities that don't add to the final product. By making these wastes a focus of our effort, we can get a huge increase in productivity.

**Kanban**

Anyone that has used a Scrum has used a simple Kanban system. In Japanese the word Kan means "signal" and "ban" means "card". Therefore Kanban refers to "signal cards". Kanban represents a "pull system" for work.

A Kanban card is a signal that is supposed to trigger action. You can see Kanban everywhere. The next time you order a barista drink at Starbucks you can see a Kanban system in place. The coffee cup with the markings on the side is the Kanban! (1)

With Scrum, a team member "pulls" a card across a board on a daily basis as accomplish work. No one is pushing the work at them in at a predefined rate.

We can employ some of the practices of Kanban that are not used in Scrum to visualize a complex production stream and allow us to apply lean principles to make that production stream as effective as possible.

**Heijunka Boards**

A Heijunka board represents the Kanban systems that use cards that represent the capacity and workflow across a value stream.

As mentioned, when you use a Scrum Task board, you are using a very simplified Heijunka board that represents a three or four stage value stream. We can expand this in production by adding the steps to our level production value stream example:

![Heijunka Board Diagram](image)

Now we have eight states for each level to be in, which represent the six stages of our value stream, and states at either end that contain levels not started or levels completed. This rotates the typical tasks of the stream into states on our Heijunka board.
This board communicates the flow of work in a level production stream more clearly than a Scrum task board.

**Applying Lean Principles**

Now that we've represented our value stream for level production, we can begin to apply some of the Lean principles to incrementally improve level production. The first step we can take is to look at the cycle time of our value stream.

Cycle time is the amount of time that it takes for a level to enter from the left in script treatment to the end when the tuning pass is done. In our example above a level takes 16 weeks of cycle time. By reducing cycle time we can be more efficient:

- Faster cycle times means more productivity.
- The more frequently something comes "off the line", the more we can fix problems with the production line.
- We can identify waste more easily and address it.

There are a number of ways to reduce the cycle time. The first way is by finding ways to reduce the size of things in the stream. For our level production example, we can do this by splitting levels up into sections, or "zones".

Each zone takes approximately 12 days to pass through the value stream -- as opposed to 16 weeks for each level. As a result, our Heijunka board looked like this:

![Heijunka Board Diagram](image)

A Heijunka board shows the progression of a zone through each stage of the value stream. In the example above, Zone 1 has stepped through each stage and been handed off to the final Tuning Pass stage at the end. This board represents a perfectly balanced flow of work across every step of our value stream.

It won't initially happen this way. There will be gaps that appear in some columns and pileups of work in others, but that's the point of doing this: those problems will be visible as soon as they occur. When we achieve transparency, we can start to fix the problems that we clearly see.

**How to Improve the Flow**

Now that we have a Kanban up and running, we have to work on making it as effective as possible. The Heijunka board will show us daily where there are gaps or pile-ups in our flow. We not only want to keep things balanced, but we want the flow to be as quick as possible.

In our example, if each zone takes 12 days to pass through the entire stream, we want to find ways of reducing that as far as we can without sacrificing quality below the level the customer will accept.

There are a number of tools we can use to improve the flow:

- Time-boxing
- Leveling workflow
- Reduce waste
**Time-boxing**

Time-boxing is something that every developer using Scrum recognizes. A Sprint is a two to four week time-box. We hold firm on the amount of time in a sprint and only vary the functionality we can deliver. The benefit of this is to create a predictable heartbeat of value being added to the game.

In production we take this a step further. We start time-boxing each stage of the value stream. For example, we might give the audio designers 10 days to add audio to a specific zone. This is different from tasks in Scrum where the audio designer would estimate their own work and tell the customers what they are willing to commit to.

In production this changes, because we have learned in pre-production how long audio design for a zone should take. Quality becomes the variable you control with time-boxing assets. We are not forcing artists to meet a set quality with a fixed amount of time.

The input is the time-box (which is the cost we are willing to pay for the asset). The output is quality that the artist is able to provide within the constraint of time.

The key to time-boxing assets is to find the correct time box size. If you choose too short of a time-box, then quality will suffer. For example, if the time-box for hi-res level geometry is set to one day, the artists would give us a level filled with untextured cubes! This would be a lower quality than what the customer wants.

On the other hand if the time-box for the zone were two months, we might end up with a zone with intricately detailed geometry everywhere. It would be absolutely beautiful, but that beauty would come at too great of a cost to the customer. It’s the job of the customer (the product owner in Scrum) to be responsible for the Return on Investment (ROI) for the production assets being created.

The product owner has to consider what the player expects from the assets in the game. When I was working on driving games, I would tell our artists to focus on making "90 mile-per-hour art". The quality bar should be set based on what the player sees when they are driving at full speed. If we sink 40 hours into creating a picture perfect fire-hydrant, the extra cost would be wasted on the player passing it at 90 miles per hour!

The product owner must keep the cost/value curve in their mind at all times:

![Cost-Value Curve](image)

This shows that the value to the customer is not a straight response to the cost of creating the asset. When you spend too little on an asset (e.g. cube city) the value to the customer will be too low. The driver might notice yellow cubes on the side of the road pretending to be fire hydrants.

Beyond a certain cost, the ROI will diminish (e.g. 1000 poly fire hydrants in a racing game). We are not relating quality to cost. We are relating value to the customer (player) to the effort we spend. We don't want to "deliver caviar to customers who want Big Macs".
Leveling Workflow

Time-boxing allows us to employ a very powerful aspect of Kanban. The cards in each column represent capacity for each stage of the value stream. As we see above, each stage can only handle one zone at a time. That is the capacity of each stage, if we have one person working at each stage.

Time-boxing is the first step in beginning to find a balanced flow for our value stream as visualized on our Heijunka board. However, one problem exists. Each stage of effort in the stream will require a different length time-box. This can cause gaps and pileups.

For example, if our level designer can lay out a level in a week, but the high res artist requires two weeks, then a lot of work can pileup for the high res artist. Conversely, if the concept artist requires two weeks to complete the concept art for each zone, the level designer might be waiting for work with nothing to do:

We have to find ways to balance this workflow smoothly so that everyone has work to do every day. One way of doing this is to balance the effort on each stage to achieve the same flow through the system.

For example, if we want to get a zone through the stream every 10 days, we start be looking at the time-boxed effort for every stage for each member of the team working on each stage:

<table>
<thead>
<tr>
<th>Stage</th>
<th>People days per zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>5 days</td>
</tr>
<tr>
<td>Concept</td>
<td>10 days</td>
</tr>
</tbody>
</table>
Level design 20 days
Hi-res art 30 days
Audio design 10 days
Tuning pass 7 days

The concept art and audio design zone takes 10 days, which is perfect for our cycle time of 10 days per zone. However the other stages have different times. Scripting and tuning take less than 10 days per stage. The script writer might have to help two teams. The designer who does the tuning pass can help out with some of the level design tasks or even testing.

For the stages that require more than 10 days per zone, we need to start adding people in parallel. For example, we would add a second level designer. The two levels could then effectively finish one zone every ten days.

Since the hi-res artists require 30 days per zone we might have to have three total hi-res artists to balance our flow. There are three different ways to add people to help out:

1. Have multiple hi-res artists work on the same zone simultaneously.
2. Break up the hi-res stage into a more detailed specialized flow (e.g. texture artist, prop artists, static geometry artist).
3. Have multiple hi-res artists working on multiple zones in parallel.

Any of these solutions will work under different circumstances. Solution number one wasn't best because our level editing tools did not support simultaneous editing on the same zone. Solution number two wasn't best because the specialized flow wasn't balanced.

We already had most of our textures and props finished. We chose solution number three. Each hi-res artist still required 30 days per zone, but the work was staggered to allow hi-res zone work to be finished every 10 days.

**Our Heijunka board then looked like this:**

Each person has one zone of capacity. By adding level designers and hi-res artists, we can add more cards per stage because we have added more capacity.

We have now established a clock rate that is the same for every stage of level production. This clock rate (10 days in our example) is called the "takt time". By maintaining and even improving our takt time across the entire stream we can level production and effect real improvements as we address waste.

**Reducing Waste**

We might be happy stopping here. We have a balanced and predictable production pipeline in place. Many developers would enjoy getting here. However the tools of lean production allow us to take things a lot further!

The first principle of Lean is to reduce waste. We have addressed many of the wastes identified by lean in setting up our Heijunka board, but I want to highlight some of the others that particularly to game production.
Many of these wastes can be identified and corrected by the team itself. This is the ideal way to eliminate waste. The main tool for this is the time-box. The time-box will exert subtle pressure on the team to find ways to become more effective. In our example above, we identified a cycle time of 10 days and balanced the entire stream to achieve this.

What happens when the product owner challenges the team to reduce their cycle time to 9 days? Will we lose 10% of the value of the assets? Surprisingly we don't! The first reaction to tighter cycle times if for the team to remove inefficiencies in how they work.

Let's use a real world example. One project in production had 10 day cycle times for the zones in their level. When they wanted to reduce that cycle time by 20% they exposed some bottlenecks.

The biggest bottleneck was the concept art stage. They only had one concept artist available and that artist was sitting across the building with the other concept artists. The concept artist took 10 days to create a dozen drawings for each zone. There was no way to get these drawings any faster.

In team discussions it turned out that the level designers and hi-res artists didn't really need all these drawings. Because the concept artist was separate from the team, much of the concept art was based on wrong assumptions. The concept artist didn't like hearing that much of his work was ignored.

The solution was to move the concept artist next to the level designer and hi-res artists. This allowed them to discuss the layout and quality bar of the work being done. As a result, there were far fewer drawings that needed to be created and the quality of the final product actually improved.

This is an example of the waste of handoffs (one of the seven wastes defined by Lean). Documentation is a major source of handoff waste. Documentation has its place for recording knowledge, but it's not a replacement for face-to-face conversation. By applying this practice to every handoff in the stream, the team was able to apply similar time savings across every stage. The team seating area ended up resembling the arrangement on the Heijunka board itself.

In the example above, the team went from producing a level every 16 weeks to producing a zone every week. With seven zones per level, the ultimate improvement to level production was 56%.

### Improving Quality

The focus on quality is a principle of Lean Production. Lean Production minimizes the inventory of unfinished work between each stage of a value stream. This allows change to be introduced much more frequency because the debt of unused components is keep low.

This is key to how companies like Toyota have improved the quality of their cars and dominate the market. If you find that you have a defect in a part of your car in production, it becomes a lot easier to introduce a new improved part when you don't have a million of the old parts sitting in a warehouse.

This philosophy translates to the assembly line as well. On the Toyota factory floor, if any assembly line worker sees a defect in the cars being produced they hit a big button nearby. If that problem isn't fixed in the next 20 minutes, the entire factory assembly line comes to a halt until the problem is fixed. This dedication to quality is unmatched in any other company and it is made possible by using Lean Production principles.

Additionally, as we reduce the cycle time by finishing batches of product, we improve our iteration cycle. Using our example, we can play finished zones every week as they "roll off the line".

We don't have to wait 16 weeks for entire level to be finished to play it. This one week cycle allows us try the level much sooner and introduce change before we have spent much time creating the remaining zones of the level.

If we build all of our levels in parallel and don't discover problems until 90% of the work is done (such as rendering or memory budgets or gameplay quality) we are faced with having to throw out a great deal of work or ship the lower quality work to meet the deadline. It's the same problem that car companies face when they decide to throw out the million potentially defective parts in the warehouse.

### Outsourcing

Outsourcing has its benefits and place in asset production. However many studios have found that outsourcing limits the amount of iteration that can take place in the creation of large assets such as key characters or levels. This limited iteration can impact quality or introduce expensive rework that limits the cost benefits of outsourcing.

Lean Production evolved to work with external suppliers. These are indispensable for manufacturing industries such as car
production. Suppliers to Lean Production companies have to become Lean themselves. The key difference with Lean suppliers is that they deliver smaller batches of parts to the main production line. This is done to allow quality improvements to be introduced frequently and at lower cost.

How can this translate to game production? With our example, we don't want to outsource the entire value stream. They key is to outsource parts of the value stream that don't require the high skill level that you want to retain in house. Generally in level production, you want to be able to keep large the layout tasks within the studio and outsource the parts that are used in layout.

Examples of this are environment sets or collections of assets that are common throughout your level. If you were creating a large city level, you would outsource all the props such as light posts, mailboxes, vehicles, building components, ambient sounds, etc. These environmental sets are brought into the layout stages (hi-res art and audio layout). This allows for continued iteration of the layoutes in the studio where they belong.

The value stream for our outsourced level production would now look like this:

![Value Stream Diagram]

The outsourced assets can be identified in early level concept development to allow for sufficient lead time for outsourcing. Many layout tools support asynchronous introduction of outsourced components. An example is the Unreal Engine 3 editor. The packaging system allows for proxy assets that can be replaced one at a time which automatically replace all instances of the asset throughout the game.

What if the Rest of the Team is Still Using Scrum?

During production, not all iteration is useless. The team is still innovating the game in areas that don't affect production. Sprints are still valuable for these teams. How do these teams work with the teams using Kanban?

Scrum teams can still use Kanban to drive production. If we have one week cycle times inside of four week Sprints, the production teams can still show the result of four cycles every Sprint review. The production team won't have to do Sprint planning, but they should still conduct regular retrospectives. Additionally Daily Scrums are still a useful practice for the production teams. Impediments will still arise that need to be addressed by the team.

Conclusion

In using this step-by-step process for creating level assets we had taken the cost of producing a single game level from 16 weeks to seven weeks (seven zones times one week per zone). This represents a 56% improvement in the cost of creating a level and required very little new tools or technology to achieve.

This is just a start. Lean production demonstrates improvements that can replace or supplement the need for outsourcing large key assets that a studio may prefer to keep in house.

Like Scrum, the adoption and practices of Lean are subject to change based on your own environment. The key is to achieve visibility on the process and to create metrics that are meaningful. Time-boxing asset creation can be a challenge. Most artists don't consider quality a variable in their work. It's a key role of the product owner to communicate their vision for not only the final product but the responsibility for ROI that they control.

References

1 [http://leansoftwareengineering.com/2008/05/21/coffee-cup-kanban/](http://leansoftwareengineering.com/2008/05/21/coffee-cup-kanban/)

I consider the two books by Tom and Mary Poppendieck to be the bibles of Lean Development: Lean Software Development: An Agile Toolkit and Implementing Lean Software Development: From Concept to Cash
I also pay close attention the to writings of Corey Ladas and David Anderson of Modus Cooperandi for inspiration and knowledge about Kanban: http://www.moduscooperandi.com/

(c) 2008 Clinton Keith

Title photo by Gregg, used under Creative Commons license.