APPLYING LEAN PRINCIPLES IN SOFTWARE DEVELOPMENT PROCESS – A CASE STUDY

Jeff Widman, Western Washington University, jeff@jeffwidman.com
Stella Y. Hua, Western Washington University, stella.hua@wwu.edu
Steven C. Ross, Western Washington University, steve.ross@wwu.edu

ABSTRACT

Lean principles have been effective in reducing waste in manufacturing organizations. These principles are now being applied in software development processes. This paper describes lean thinking concepts and the process of applying lean in the technical coding at IMVU - a virtual worlds company. It also summarizes lessons learned in the implementation of lean for software development processes.

Keywords: Lean Principles, Software Development, Case Study, Virtual World

INTRODUCTION

Based upon the Toyota Production System (TPS), lean thinking is a systematic methodology for identifying and eliminating waste in manufacturing and administrative processes through continuous improvement by flowing the product and service at the pull rate of the customer. James Womack and Daniel Jones, who developed the concept of lean after studying American and Japanese automotive production [8], define lean as the process for doing more with less and less – less human effort, less equipment, less time, and less space – while coming closer and closer to providing customers exactly what they want [9]. While lean has been applied in manufacturing organizations since its inception, lean thinking techniques are also relevant and effective in non-manufacturing environments. This paper discusses lean thinking concepts and the process of applying lean in the technical coding at IMVU, a virtual worlds company.

Lean Thinking Principles

As described by Womack and Jones [9] and Tapping and Shuker [7], lean management relies on five major principles, and the success of these principles rely on lean champions and employee involvement and empowerment. The first principle requires that an organization define value from the customer's perspective and determine what activities customers willing to pay for. Next, the organization must identify and map the selected value stream and eliminate wastes. A value stream is the entire set of activities required to design, order, and make from raw material to the finished product for a specific product, product family, or service that seeks to eliminate waste and optimize the productive system in order to satisfy the final customer [9]. Womack and Jones define seven major wastes: overproduction, inventory, waiting, defects, processing, movement, and transportation. Third, the organization should develop pull systems for moving work through the value stream according to the rate of customer demand. Fourth, employees have to be involved and empowered for continuous improvement. Fifth, an organization must sustain its lean initiatives and strive to pursue perfection. Our analysis of the case relies on these five principles … other authors have developed similar sets with varying numbers of principles.

Software Development Processes

Compared to manufacturing process, software development is very different as it is not rolling out a physical and tangible product. Instead, it is creating an intellectual property, which is highly dependent on the developers' human qualities such as creativity and efficiency. In a way, "everybody's head has to be in the game" [2]. Thus, software development process has more uncertainties and complexities than manufacturing process. In fact, it is usually chaotic, and constantly changing with coding, bugs, and testing, overlapping with each other.

Mary and Tom Poppendieck are considered early adopters of lean concept in software development process. Their book on agile software development [4] present a set of lean thinking tools for software development. In the sequel published six years later, Mary and Tom Poppendieck [5] proposed seven principles that put main focus on people and communication during the implementation process, including technical leadership, knowledge building, and decision making. They also included 22 tools for identifying problem areas and discovering possible solutions.

With software techniques changing every two or three years on average, Middleton and Sutton [3] illustrates the benefits of lean production in embed-

ded software projects. They focus on finding the best combination of lean techniques and software technologies such as software reuse, SEI CMM (Capability Maturity Model from Software Engineering Institute), and extreme programming (XP). Regardless of the type of software technologies, they believe that "a company that builds on the foundation of lean production won't be competitively pressured to completely rethink and rebuild its processes every time another 'latest technological breakthrough' comes down the pike" (p. xxiii).

Besides the people and technical aspects of software development, software companies often struggle with the overall product development process. It is the balance of three things: product features, quality, and shipping schedule. Some features may require longer lead times, therefore, delay the shipment of the final product. A series of releases is more likely to yield the software that customers desire, but shipping the product early could leave devastating bugs in it. Therefore, producing the right product, at the right time, and at the right price, is even more challenging for software companies. Thus, in this paper, we focus on the benefits of lean principles on software development process in dealing with product features, quality, and delivery.

CASE STUDY

What is IMVU?

IMVU Inc. (www.imvu.com) is a virtual company where users meet as personalized avatars in 3D digital rooms. Founded in 2004, IMVU has 25 million registered users, 100,000 registered developers and reached $1 million in monthly revenue. Over 90 percent of IMVU's revenue is from the direct sale of virtual credits (a form of currency) to users who purchase digital products from its 1.8 million item digital catalog. IMVU has won the 2008 Virtual Worlds Innovation Award and was also named a Rising Star in the 2008 Silicon Valley Technology Fast 50 program. IMVU receives funding from top venture investors Menlo Ventures, Allegis Capital and Bridgescale Partners. Its offices are located in Palo Alto, CA. (http://crunchbase.com/company/imvu)

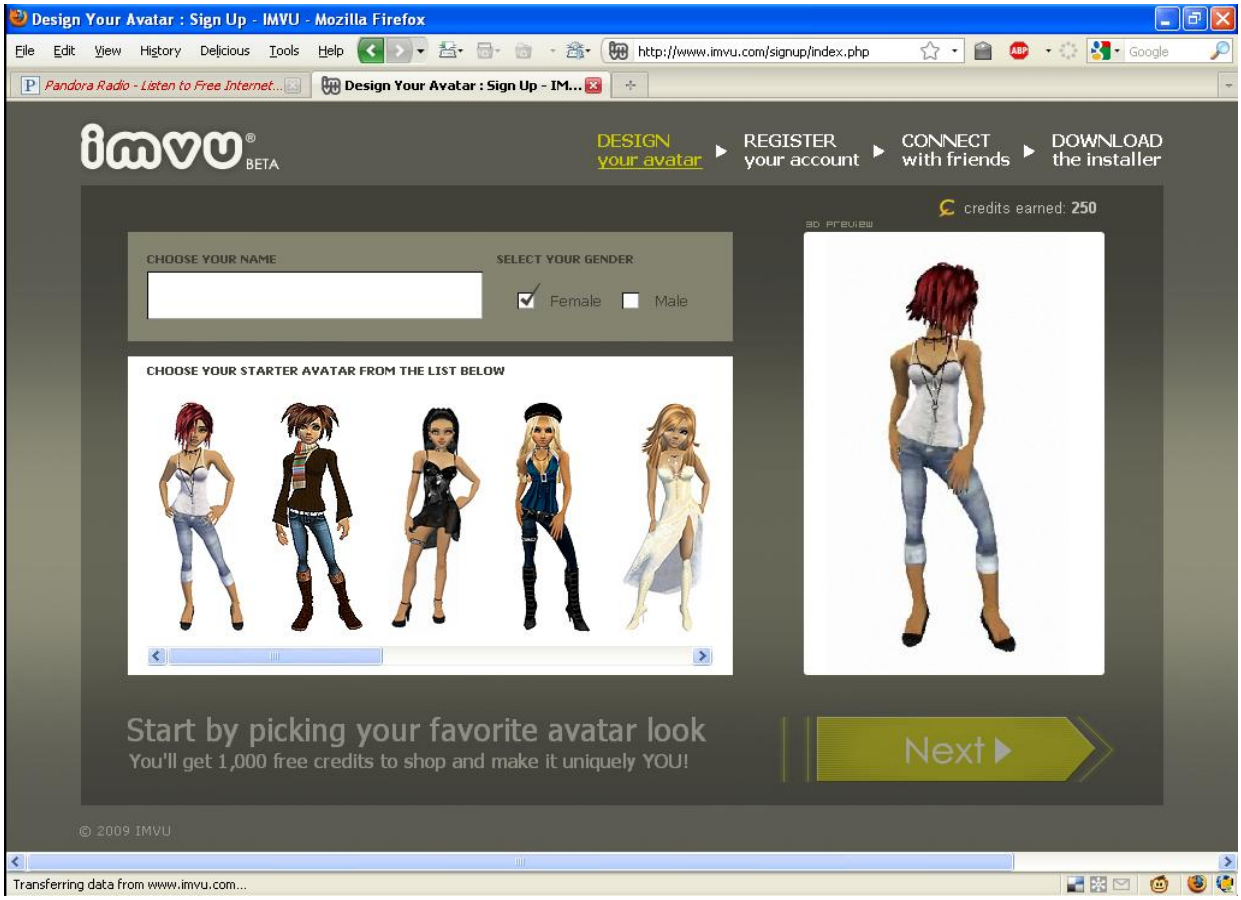

Figure 1. Design Your IMVU Avatar

Software Development at IMVU

IMVU's founders had previously founded There.com—a virtual world startup that took three years to build, burned through a ton of money, and was an abysmal failure after launch. However, from an engineering perspective, There.com was an amazing success, as they built it ahead of schedule, maintained tight quality standards, and solved multiple difficult technical problems. Still, it wasn't a commercial success—and large amounts of time and money were wasted. As a result, IMVU's founding team decided to build the minimum viable product and then test it with users—even if the product seemed only half-built (an engineer's nightmare).

As a result, IMVU was one of the startups that pioneered the "build-just-a-little-and-get-customer-feedback" model. This model was only possible because of the application of several lean principles at the technical level in the development process.

Lean Principle #1: Specify Value in the Eyes of the Customer

From the beginning, IMVU's founders decided they wanted to build a culture of "ship, ship, ship." From a business perspective, this makes a lot of sense, but from an engineering perspective, it's like pulling fingernails with a pair of rusty pliers:

"*Bugs were all over the place, extremely ugly looking, and only the most rudimentary features. While I used to enjoy showing my family what I'd built, I remember thinking 'I never want my family to know I've released this pitiful of a product.' But it allowed us to test the market and get feedback, which allowed us to drop certain features that didn't resonate with our audience, and focus on others that our customers really liked*." [6]

In essence, releasing a sub-par product allowed IMVU to avoid over-production wastes by putting man hours only into features their customers liked.

Lean Principle #2: Identify Value Stream and Eliminate Waste

The IMVU team worked hard to cultivate the "ship, ship, ship" mentality. For example, on their very first day, most developers were expected to write some code and push it into production. Even though it was generally just a small bug fix or a miniscule feature, this "release-code-on-the-first-day-of-work" idea seemed revolutionary for most new hires.

Continuous deployment reduced the wastes of over-production, waiting, and processing. In a traditional development process, multiple engineers are busy building multiple features based on the last bit of stable code. When they try to deploy their feature after two weeks of work, they find that someone else deployed a different feature the previous day and the two features don't play well together. Continuous deployment allows engineers to upload their work instantaneously – thus ensuring engineers are always working from the same base code. This avoids spending extra weeks making the feature code compatible.

Lean Principle #3: Make Value Flow at the Pull of the Customer

IMVU projects have an eight week Return on Investment (ROI) target. Whenever someone suggests a small project, they are asked to provide a general roadmap showing that the project could repay the time investment in eight weeks. Projects are continuously tested on small numbers of IMVU users – who often had no idea they were part of a bucket test. If a project shows success, they keep working on it. After a few weeks, if the numbers shows the project had zero chance of positive ROI, it is shut down immediately. Over time, as IMVU matures, this project ROI target is expanded:

"…we deliberately changed these project payoff definitions over different stages of the company—trick is to flex here… as the size of your capital expands, make bigger bets and take bigger risks with projects." [1]

Lean Principle #4: Involve and Empower Employees

IMVU implemented the 5 Why's process, also known as "Root cause analysis," to involve and empower its employees during trouble-shooting processes. 5 Why's process is the technique of asking why five times to get to the root cause of a problem when it occurs.

As reported in blog posts by Ries [6], each IMVU engineer has his/her own sandbox which mimicked production as close as possible. IMVU has a comprehensive set of unit, acceptance, functional, and performance tests, and practiced Test-Drive-Development across the whole team. Engineers build a series of test tags, and quickly run a subset of tests in their sandboxes. Revisions are required if a test fails.

To keep developers on the same code before it passed the various tests, IMVU created the equivalent of a

Kanban system plus an Andon cord (automated testing and immediate rollback system). Developers are assigned a single task, and not allowed to move onto the next task until their code not only successfully passes the automated testing, but also has successfully deployed. Only then, they can pull the next task from the queue. This means that developers have a little bit of idle time while the tests were running. It also means that code is fully completed before a developer moves on. As a result, engineering is optimized for productivity rather than activity:

"*We realized we had been optimizing to keep our engineers busy, when we really cared about developer efficiency as measured by whether the product moved forward. We started measuring (total feature output)/(time in the office), rather than (coding time)/(time in the office). They're NOT the same. After implementing this new system, developers had a little extra time on their hands, but feature output increased*." [1]

Lean Principle #5: Continuously Improve in Pursuit of Perfection

The problem with all this emphasis on "ship, ship, ship", was different bugs in the code kept taking the site down. Sometimes it was simply a scaling issue—new upgrades worked fine on an engineer's computer, but crashed when hundreds of thousands of users tried it. Other times, it was a new employee releasing some feature without understanding how the previous code base worked. From a business perspective, it didn't matter what the problem was; if the site was down, IMVU was losing money.

From a technical perspective, each new problem required a different solution. Solving scaling issues is very different than solving a single infinite loop problem. The only practical fix was either cease continuous deployment or institute automated tests that checked the code, plus allowed for immediate code rollbacks if any server started to crash.

"*For several months we went back and forth on this automated testing idea. It was REALLY difficult, and required a lot of computer processing power—we had to run a comprehensive set of automated tests every time there was a code submission. As our code base grew, so did the complexity of the tests. As the number of our engineers grew, so did the frequency of the tests. I calculated that we were adding approximately one test machine per engineer every six months. It was a N^2 function where N is the number of technical employees. We kept waffling whether it was worth it, but it enabled so many things—not the least of which was continuous code deployment*." [1]

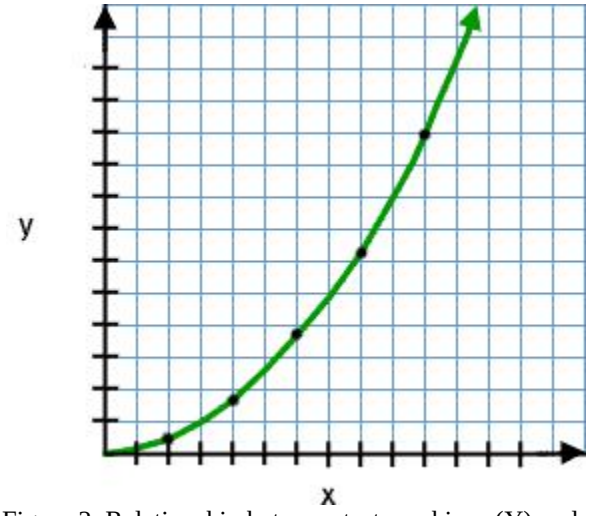

Figure 2. Relationship between test machines (Y) and employees (X)

Eventually, IMVU architected a series of automated tests that looked at every new code check-in, tested it, and then pushed it onto the live servers. If at any point the code crashed—either during testing or once it started running in the wild—the automated tests instituted a rollback to the last verified good version, and sent a nice little e-mail back to the engineer that said "We're sorry, but it looks like your code ABC caused a problem at XYZ. Afraid we can't let your code go live until this is fixed." As a result, the automated testing caught an amazing number of errors, and IMVU management started pursuing massively high quality expectations.

"*We set a very high bar of approximately 10,000 test cases, intermittent test with 100,000 assertions, and we were pushing code twenty times per day—so our quality rate was 1:2,000,000. It took us about ten minutes to run all these tests*." [1]

CONCLUSIONS

IMVU successfully implemented lean principles at the technical level in the software development process. They encountered many common challenges that software companies face: choosing the right product feature, long development cycle, endless testing and debugging. IMVU found solutions by sticking with the basic lean principles. They were able to identify and reduce common wastes in software development process, specifically, overproduction, waiting, process, and defects.

IMVU clearly demonstrated the importance of lean implementation in the software development process. The implementation of lean principles cannot turn software development into a production line environment, with scientific methods for each step of the way. However, it can help turn a chaotic, constantly changing process, into a much more predictable, fast moving, and streamlined process. Lean implementation coupled with brilliant designs and fully engaged intellectual team can help deliver great software products.

It would seem that the rapid release cycles called for by lean principles can only be effective if there is a comprehensive and rigorous testing environment. An interesting question is whether IMVU's practices (such as daily release online) would be applicable to software companies that focus on packaged rather than online products. In this case, the "customer" is a combination of the other developers and the ultimate consumer. IMVU's experience challenges the conventional wisdom in software development. Can it be beneficial to all software companies striving to deliver the right product, at the right time, and at the right price? Middleton and Sutton [3] believe that the benefits work across different types of software. Yet, they also recognize that lean software is far too early in its evolution. Further research is needed to better define lean software development paradigm.

REFERENCES

1. Hondl, Chris (2009). Personal Interview. San Francisco, CA.
2. McCarthy, Jim (1995). *Dynamics of Software Development. Redmond*, WA: Microsoft Press.
3. Middleton, Peter and James Sutton (2005). *Lean Software Strategies*. New York, NY: Productivity Press.
4. Poppendieck, Mary and Tom Poppendieck (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional
5. Poppendieck, Mary and Tom Poppendieck (2006). *Implementing Lean Software Development: from Concept to Cash*. Addison-Wesley Professional.
6. Ries, Eric (2009). Online blog http://www.startuplessonslearned.com/2009/07/how-to-conduct-five-whys-root-cause.html.
7. Tapping, Don and Shuker, Tom (2003). *Value Stream Management for the Lean Office*. New York: Productivity Press.
8. Womack, James; Jones, Daniel; and Roos, Daniel. (1990). *The Machine That Changed the World*. New York: Simon and Schuster.
9. Womack, James and Jones, Daniel (2003). *Lean Thinking: Banish Waste and Create Wealth in your Corporation*. New York: Free Press.

ADDITIONAL SOURCES

Fitz, Timothy (2009). Continuous Deployment at IMVU, Doing the impossible fifty times a day. http://timothyfitz.wordpress.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

IMVU Corporation Engineering Blog (2010). http://engineering.imvu.com/

Brainhuddle.com (2010). Startups Lessons Learned: Case Studies. http://brainhuddle.com/blog/?tag=imvu

Impact Lab (2010). Many Young Internet Businesses Are Using the 'Lean Start-Up' Principles. http://www.impactlab.com/2010/04/25/many-young-internet-businesses-are-using-the-lean-start-up-principles/

Shaping Software (2009). Patterns and Practices of Lean Software Development. http://shapingsoftware.com/2009/06/22/patterns-and-practices-of-lean-software-development/