Learning from Lean
Kent Beck, Three Rivers Institute

Summary

Both Lean Production and Lean Product Development have lessons for software developers. This paper examines one principle, waste, shows how it is expressed differently in production and product development, and how those lessons translate to software development.

Introduction

I first read Taiichi Ohno's *Toyota Production System* on a trans-Atlantic flight on my way to XP 2002. As the water flowed far beneath our wings I thought (and, judging from the reaction of my seatmates, said), "Right on! Exactly! You tell it!". Once at the conference, Professor Zaninotto gave a keynote that was a model of clarity and depth relating the economics of lean production to the economics of software development (I couldn't find the slides to reference), reinforcing my sense that the ideas Ohno described could be instrumental in improving software development. But which ideas?

The aspects of the Toyota Production System (TPS) that first appealed to me were those that focused on the social purpose of work. Software development often seems to occur in a purely technical context devoid of business or social purpose. Questioning why the code I am writing today is good or valuable helps me focus my development and avoid wasted effort.

When I got to Alghero for the conference, I started talking about Ohno and TPS. I immediately encountered Tom and Mary Poppendieck. I didn't know them except by reputation at that time but I found we agreed on the importance of the social message in TPS. It was when we started to explore the practical implications of TPS for software that I got a shock.

I explained that software development was like a lean production line. The repetitive "standard work" consists of tasks like writing a test, refactoring, planning, and building. TPS suggested ways of smoothing and accelerating these processes.

"No, no," Mary said, "The real lessons of TPS for software are in the way Toyota develops new products, not the way it manufactures." When Mary speaks, it's a good idea to listen. In this case, though, I wasn't convinced.

Five years on I think I have a more balanced perspective. I am right that lean manufacturing has valuable lessons for software and Mary is right that lean product development has valuable lessons for software. In this paper I intend to show how the same lean production principle can offer different but valuable lessons for software when seen through the lenses of manufacturing and product development. The key is the persistent application of consistent principles.

A full explanation of TPS as applied to software would take books. This paper is limited to exploring the practical implications of TPS for software that I got a shock.

Waste

One key principle of TPS is the elimination of waste. Rather than focus directly on increasing productivity, in TPS productivity is a trailing indicator: get rid of waste and productivity will rise as a consequence.

This principle plays out differently in manufacturing and product development. On the production line, Ohno lists seven wastes, headed by the waste of over-production. Each item created has two sources of value: the intrinsic value that someone else will pay for the item and the information value, feedback about the making of the object. Over-production creates the illusion of value by making more things that are potentially salable, but by not using or selling them immediately the information value of the items evaporates. If an upstream machine needs adjustment but the parts it produces aren't used immediately, no one will notice the problem
until much later when many more defective parts have been created. Producing only what is needed when it is needed gives you the information necessary to run production smoothly.

Traditional software development is rife with over-production: too many requirements to implement this release, too much architecture for the current needs of the system, too much code to be adequately tested, too many features for customers to use. Most of this excess work is done, ironically, in the name of efficiency, but by producing more than we can consume we lose the information value of our work, concealing the information needed for large-scale improvement.

TPS has a large and sophisticated set of techniques for throttling production back to actual use. Many of these have analogs in Extreme Programming. Practices like Test-Driven Development encourage developers to code to today's needs, not uncertain future requirements. If the requirements and design aren't done first, their work must be spread across development, a shift which is supported by incremental design and the social practices that help the team maintain a shared understanding of the system.

That's waste as applied to software through the lens of manufacturing: produce only what you can consume and use the information generated for improvement. Toyota has been careful to apply the same principles to all aspects of its business, while recognizing that different contexts require different application. Eliminating waste is a key principle in Toyota's product development process. However, in product development you deal with a wide range of problems and tasks. Over-production would be wasteful, but in practice it is not the biggest waste in product development. The biggest waste in product development is re-learning.

Product development teams are large and diverse. When someone on the team learns something, that knowledge needs to spread quickly to the whole team. Otherwise, someone else could learn an opposing truth and the two people could begin working at cross purposes. The re-learning process can also span time. What one part of the team learns earlier in the development process could become important to another part of the team later. Spending time and money learning the same thing multiple times is a waste of opportunity and a potential source of contradictory decisions.

Some of the techniques used to promote efficient learning at Toyota are also found in Extreme Programming, like the "try it and see" attitude. Others, like the appointment of Chief Engineers, are not common on XP teams.

So here is a single principle, eliminating waste, that brings different insights to software development depending on the metaphor. If software development is lean manufacturing, software development should aim to produce decisions at the same rate as it consumes them. If software development is lean product development, software development should strive to retain and disseminate learning. Both lessons can positively influence software development.

Conclusion

The lessons of TPS can be applied to software through both metaphors--software development as product development and software development as lean production. Which metaphor is most helpful depends on perspective. I encourage you to try them both and see what you find.

http://www.threeriversinstitute.org/LearningFromLean.html