



12 Things to Shorten Your Lead Time

Stephan Schmidt

BETA

Early Access

This eBook will give you insights into how to **reduce the lead time of your software development process**. In many companies the perception is as follows: why are we so slow? Marketing probably asks you why you have lead times of several months as an IT-department, from their ideas to seeing the feature live. Many companies have large lead times, often unnecessarily large. The good news is: You can shorten them, sometimes by several hundred percent!

Why shortening lead time?

Shortening lead time is possible, but not easy. So why should you do it? There are many reasons to shorten your lead time. The sooner you get a feature live on your platform, the sooner it will generate money for you. Shortening the lead time from three months to one month means, the feature will earn two additional months of sales. This is the case for every feature that goes live earlier. More money earlier means more money to invest - more investment means a larger market share sooner.

Short lead times might mean the difference between being a first mover or not. There are lots of first mover advantages, if you're faster than your competition, your customers will perceive you as the market and innovation leader, not the copy cat. Short lead times differentiate the good from the great.

When going short, you will need a lot of discipline. As we'll see shortening lead times means you need to solve many problems that will arise in your process. No more slack. Solving those problems will make you lean and give you cost advantages over your competition.



What is lead time?

"A lead time is the period of time between the initiation of any process of production and the completion of that process. Thus the lead time for ordering a new car from a manufacturer may be anywhere from 2 weeks to 6 months. In industry, lead time reduction is an important part of lean manufacturing." **Wikipedia**

When measuring lead time, software development and IT departments often start the clock when they start to work on a feature. But this is much too late. Take the perspective of your customers. It's necessary to start the clock as soon as your customers in marketing, sales, accounting, back-office have an idea and issue a request.

When does the clock stop? Many development processes stop with the last line of code written or the last feature tested. As before perceive the process the way your customers do. The endpoint of feature development is when the idea generates money. This often means when it's deployed to the live system, not earlier.

To measure the process from the point of view of your customers, do not optimize locally, optimize the whole process. The process is done when the idea generates value for your customer; it's not done at the end of development or the end of QA.

But how to shorten your lead time? This eBook is a step-by-step guide towards shortening lead time. The internet is full with information, blog posts, articles or tweets. Many books exist on the topic of lean production and lean development and some good ones are listed in the sources section. The problem is not to know how to do it, but what to do when. With all this information, what shall I do?

The overall goal of all actions for reducing lead time is reducing waste. Your process is clogged with waste, called muda, mura and muri in the lean community. The most common waste in software development is buffers. When features stay for a long time in a buffer before design, or before development, you are creating waste. Other muda wastes in software development are partially done work, extra features [1], relearning, handoffs, task switching, delays and bugs. Other waste is muri, which means overburden and mura for inconsistencies or unnecessary variation.

1. Measure, measure, measure
2. Increase quality
3. Reduce rework
4. Frequent releases
5. Stop working in parallel
6. Shorter stories
7. Visualize and manage flow
8. Rigorously cancel meetings
9. Continuous deployment
10. Shorten product management
11. No single point of failure or bottleneck
12. Leveling work

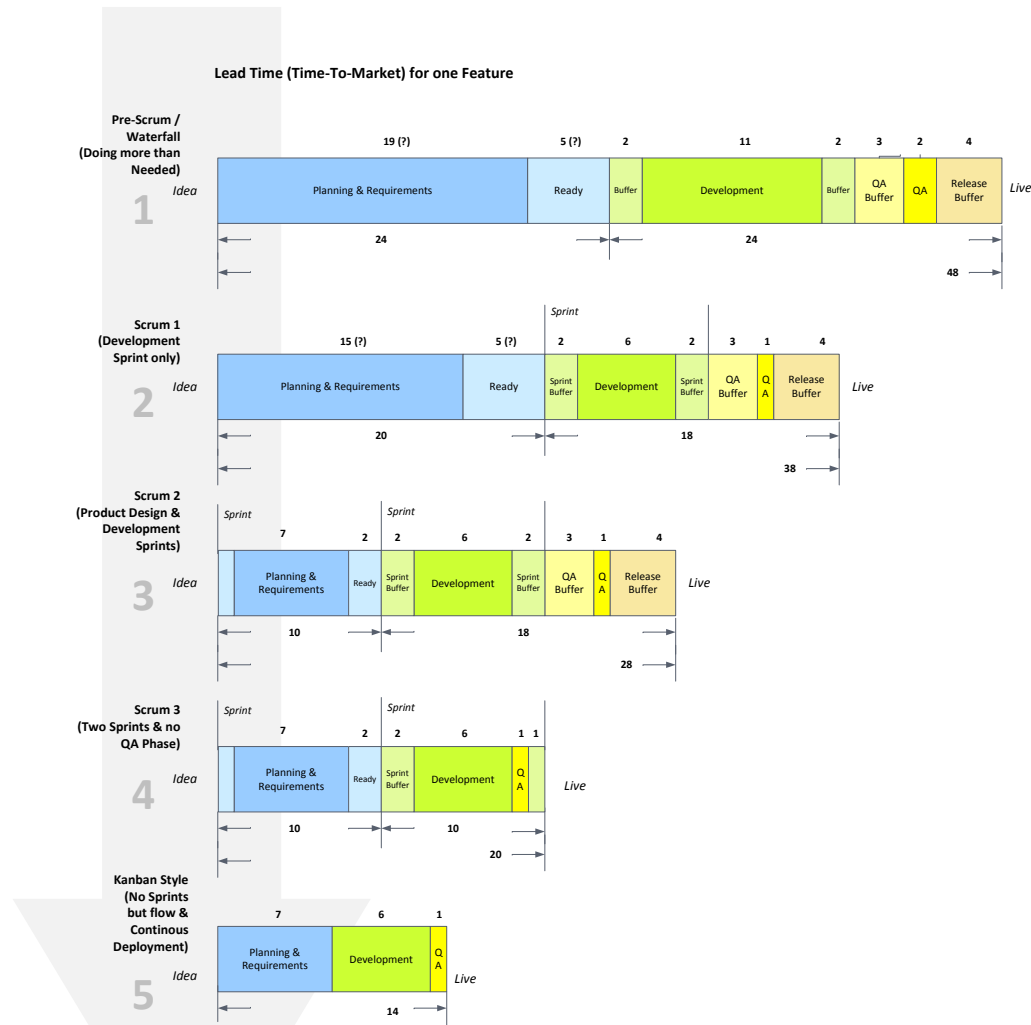


Figure 1 – Reducing lead time. With subsequent technical measures (Scrum introduction, optimizing product management, dropping an inspection after the fact QA phase and continuous deployment) lead time can be reduced drastically from 48 to 14 days in this example

11 Actions you can take now:

1. Measure, measure, measure

The very first thing is to start measuring your process if you don't already do so. For starters record the day when the idea was filed, when it was ready for development, when development started, when development ended, when it went into QA, and how long it waited to be released. Armed with those numbers you can calculate your lead time from idea to going live. You can calculate the duration of your buffers: how long a feature is waiting before going into development, before going into QA and waiting for being released. Without measuring, all other attempts to reduce time-to-market are futile.

| Sprint | Story | Description | SP | Feature Idea | Dev Ready | Start Dev | End Dev | Pre-Dev | Buffer Before | Time Dev | Buffer QA | Wait to REL | Lead Time |
|------------|------------|-----------------------------|----|--------------|------------|------------|------------|---------|---------------|----------|-----------|-------------|-----------|
| 22.06.2009 | 03.07.2009 | "Make things work" | | Release 42 | 20.07.2009 | | | | | | | | |
| 12 | S1 | Reimplement PayPal Provider | 2 | 02.06.2009 | 10.06.2009 | 22.06.2009 | 29.06.2009 | 6 | 8 | 5 | 5 | 15 | 34 |
| 12 | S7 | Add customization | 8 | 04.06.2009 | 15.06.2009 | 22.06.2009 | 01.07.2009 | 7 | 5 | 7 | 3 | 13 | 32 |
| 12 | S12 | Add REST Interface to API | 40 | 15.06.2009 | 16.06.2009 | 23.06.2009 | 02.07.2009 | 1 | 5 | 7 | 2 | 12 | 25 |
| 12 | S10 | Evaluate Cassandra | 2 | 26.05.2009 | 20.06.2009 | 23.06.2009 | 24.06.2009 | 18 | 1 | 1 | 8 | 18 | 39 |
| 12 | S8 | Migrate Backend to Scala | 13 | 11.06.2009 | 23.06.2009 | 30.06.2009 | 03.07.2009 | 8 | 5 | 3 | 1 | 11 | 27 |
| 12 | S4 | Implement Password Reminder | 13 | 16.04.2009 | 22.06.2009 | 22.06.2009 | 29.06.2009 | 47 | 0 | 5 | 5 | 15 | 67 |

Table 1 – Example measuring feature metrics in Excel. This calculates the lead time, development time and several buffer times

2. Increase quality

Low quality is one of the biggest wastes in development. The goal with increasing quality is to drop your QA phase altogether. In most companies there is a dedicated QA test

Low quality is one of the biggest wastes in software development.

department and a test phase after development has finished. Often it takes days or weeks before a feature goes through QA, increasing your lead time significantly. Moving the testers to your developers, making them test each feature as soon as a developers says she's finished and before the code is stored into your source repository, reduces turnaround time for features and bugs dramatically and allows you to drop the QA phase. To do this, you probably need to increase your quality a lot, which means

- adding testers to teams,
- educate your developers,
- set a zero-bug policy,
- add automatic tests on unit level, on regression level, for functional testing and integration testing.

Increasing quality dramatically will help you later on the road to continuous deployments (0 day release buffer), see below.

3. Reduce rework

The worst case are bugs detected by your customers. Not only will this tarnish your image, it will take a lot of time to fix.

1. The bug is filled with customer support,
2. goes to your QA department,
3. is verified,
4. is entered into a bug tracking system,
5. given to a developer,
6. verified again,
7. perhaps checked with product management that this really is a bug,
8. analyzed,
9. fixed,
10. verified by QA again,
11. closed in your tracking system,
12. packaged for a release
13. and deployed.

Beside lots of work, the developer and QA need to familiarize themselves with the feature again, which takes much more time than fixing the bug during development, detected by a test or by the developer and fixed immediately. Bugs and rework will also reduce the number of features your developers can finish, because fixing bugs needs so much of their time.

Get it right the first time.

4. Frequent releases

The biggest buffer is often the release buffer, so we'll talk about that one before all other buffers. If you have a 6 month release cycle, most of your features will be developed, tested and could earn money, but need to wait several months to be released. **You should shorten your release cycle to 1 month.** Then if you're comfortable with a 1 month cycle, shorten it to 2 weeks. What works great is to have iterating releases, every

two weeks a feature release, in-between a bug and configuration release. This is easily manageable. Going down to 1 week feature releases usually doesn't pay, especially if you're agile. Instead of reducing the release cycle below 2 weeks, better go for continuous deployments (see below). With agile development, align your iterations with your release cycle to not artificially extend the release buffer. If you have 2 week iterations in development - and not overlapping iterations between teams - use a 2 week release cycle. The release buffer can then be shortened from a maximum 120 work days to below 10.

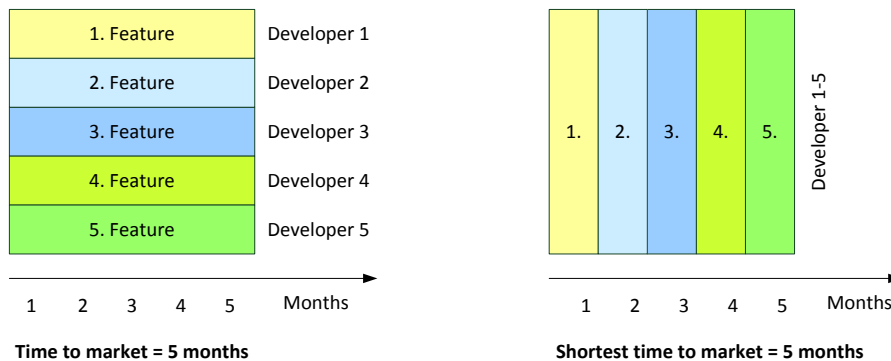


Figure 1 – Parallel versus serial development

5. Stop working in parallel

Often development teams work highly parallel, which means each developer works on his own feature. Suppose it takes each developer 5 months to finish his feature, then the time to market for every feature is 5 months (see figure 1). If instead all developers work on

one feature, and work can be parallized¹, then the first feature will be finished in one month, the last will be finished after 5 months, reducing the average lead time to 3 months.

With parallel development there are 5 unsatisfied customers, each waiting five months for their feature. In the second case, only one customer will need to wait 5 months, all others get their feature earlier online. As a positive side effect, because you only work on each feature one month instead of five, your customers will blame the waiting time on the lack of enough developers. **They will blame their waiting time on slow development in the first case.**

6. Shorter stories

As studies have shown, the 80/20 rule holds true for software development. With 20% of development you will satisfy 80% of your customer needs. It's important to identify the 20% of your stories that provide 80% of the value. Shorten the features or stories to the 20% that provide most value and your lead time for features will be dramatically reduced (in an optimal case by 5 times).

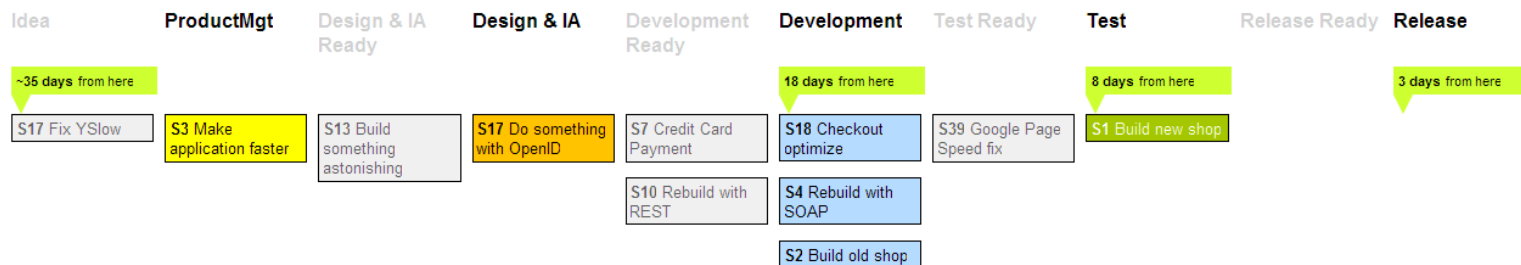
Keep also in mind, studies have shown that 64% of features are never or rarely used. Those clog your development pipeline, increase lead time for all other features and **aren't used by your users!**

¹ There are limits on how much a feature can be parallized. With low developer numbers like 5 and big enough features work can be parallized enough to gain significant lead time.

7. Visualize and manage flow

Visualize your stages of development. As a visual tool in lean development people use a Kanban board. In software development this board contains all features currently in the pipe. If a feature moves from one stage to the next, from “Development” to “Test Ready”, the card is moved to the corresponding column. As an information radiator the board tells everyone in the company in which stage a feature is. Based on that transparency, flow can be managed. It’s easy to see where congestion or a bottleneck is. More advanced techniques of managing flow are work in progress (WIP) limits. Each stage limits the number of features it can contain; **understaffed and unbalanced stages will then easily show by blocking flow.**

Kanban Board Simple Kanban



Written by Stephan Schmidt, who can be found on Twitter <http://twitter.com/codemonkeyism>. Blog at <http://www.codemonkeyism.com> using JQuery with the DragSort plugin. This software application is Twitter-Ware. If you like it, use it, or have something to say, follow me and tell me on Twitter at @codemonkeyism. The newest version can be found on www.simple-kanban.com

8. Rigorously cancel meetings

Another big waste that prevents your developers from developing code are meetings. You thought you did hire developers to develop code. But often they aren't. Rigorously remove meetings from your corporate culture. If in doubt, do not meet. Let your developers write code and **work on customer value**. Taking a look at Scrum will help. Scrum as an agile methodology has a pre-defined set of meetings, **all with actionable results and time boxed**. This minimizes the number of meetings without clear results that seem to go on endlessly.

9. Continuous deployment

With the goal to remove all buffers, set your release buffer to 0 days. This is called continuous deployment. A feature is developed, checked into your source repository, automatically tested, automatically deployed to one server, checked again and deployed to the next server - or in case of a failure the version is rolled back to the last version of your platform that worked. Continuous deployment - versionless software - reduces your release buffer to 0. Every feature is live minutes after development has finished. This needs a lot of discipline though.

You need a high degree of automatic testing and a high coverage for unit testing, regression testing, integration testing and functional testing. It helps to have first mastered tip 2 "Increase Quality" and tip 4 "Frequent releases". Those should have taught you most of the discipline that is needed. They should also have taught you to make your deployments as easy as possible and mostly automatic. Because continuous deployments need to be automatic in order to work.

10. Shorten product management

Many companies measure productivity and time to market from the start of development to the point when a feature went live. But often a lot of time is wasted before development starts. Ideas are thrown around, discussed, refined, changed and when finally after weeks or even months the feature goes into development, the customer asks "why does development take so long?" The key is to reduce the time a feature stays in product development: **Stop Talking. Start Doing.** This can either be accomplished with setting time goals and time boxes for each feature or with running iterations just like your developers do. Prevent analysis paralysis. Set a time box of one week per feature, after one week it should be ready for development. You can always improve your ideas later: During development when you're agile, with a new feature if you're not.

Product management is often prone to the same errors that plague development: working on too many features at the same time in parallel. Stop working on many features at the same time **now**, start working only on a limited set of features (called work in progress or WIP). Push a feature through product management and requirement gathering. Don't procrastinate. Have a list of items for level of done. When are you done with product management and the feature is development ready? A concrete list will give you goals (SMART) you can achieve.

11. No single point of failure or bottleneck

With a single point of failure (SPOF), your lead time becomes unpredictable. Should the single point of failure break, your lead times will go up dramatically. In software development SPOFs can be all involved where there are not enough of:

- Perhaps you only have one designer
- Or a single product manager who can use a wire frame tool
- The only developer who knows how to write that special backend code
- A system operator who is the only one who can deploy certain services
- The single database guru for your legacy system

Single points of failure are not only a risk for increasing your lead time, but often they are bottlenecks who increase the lead time of all your features. Bottlenecks can also be found in support teams, design teams, database teams who are understaffed. **When measuring buffers and lead times, it's easy to detect where you're understaffed and take measures.**

12. Leveling work

Unnecessary variation or unevenness in an operation is called mura in Japanese. Together with muri which means overburden. In software development you often have very stressful phases alternating with phases where there is a low amount of work. This not ideal, because it will burn out your developers when they are overburdened, and you pay too much for developers when there is only a low amount of work. **Lead time is becoming erratic.** The key is to leveling your work. In Japanese lean terms this is called heijunka. Leveling the work level reduces lead times for all features and makes lead time much more predictable. The best way to level work is to introduce a pull process. Later stages in the process (development) are pulling work from earlier stages of development (product management or business analysts). Contrary to pushing work downstream, this will create a leveled work load with optimal developer utilization. Otherwise you have peaks, so you need to hire for peaks.

Web Sources

The 7 Software Development Wastes - Part 1, Jack Milunsky

The Three M's - The Lean Triad, InfoQ, Roman Pichler

Books to Read

Implementing Lean Software Development, Mary and Tom Poppendieck

The Toyota Way, Jeffrey Liker

Lean Thinking, James P. Womack and Daniel T. Jones

About the author



Stephan Schmidt is head of development at brands4friends, a large eCommerce shopping club for brands. He has over 25 years of programming experience and more than 15 years of internet technology experience. He was head of development, team manager, consultant and CTO in different companies and is a speaker, author and blog writer. He specializes in organizing and optimizing software development helping medium companies and startups by increasing productivity with lean software development and agile methodologies.

BLOG: Codemonkeyism – Programming is hard (<http://www.codemonkeyism.com>)

TWITTER: Codemonkeyism (<http://www.twitter.com/codemonkeyism>)

Download this

This **eBook** is available from <http://www.codemonkeyism.com>

Credits: Creative Commons image by laihiu used under a Creative Commons license from Flickr, Las Vegas cover image by Roadsidepictures under a Creative Commons license on Flickr