Lean in the Software Test Lab

Kathy Iberle

kiberle@kiberle.com

Abstract

This paper applies the methods of Lean Science to the management of software test labs. The fundamentals of Lean Science are introduced, and the effects of batch size and Work-in-Process limits are explored. Lean Science can dramatically increase productivity and throughput in most test labs. You really can work smarter, not harder!

This paper will give you a gut feel for why these methods work and why they produce dramatic improvement. It also introduces enough of the basic concepts and tools to allow you to assess the potential for your own work.

Biography

***Kathy Iberle** has over fifteen years of experience in developing processes to plan and execute efficient development of high-quality products in a variety of fields. She has been working in agile software development and Lean development for many years, and has developed unusual expertise in the challenging area where hardware and software meet. Kathy recently retired from Hewlett-Packard after a rewarding career and is now the principal consultant and owner of the Iberle Consulting Group. Kathy served as co-chair of the Program Committee of the Pacific Northwest Software Quality Conference (PNSQC) in 2009, participated in the invitation-only Software Test Managers Roundtable for five years, and has published regularly since 1997. Kathy has an M.S. in Computer Science from the University of Washington, and an excessive collection of degrees in Chemistry from the University of Washington and the University of Michigan. Visit her website: www.kiberle.com.*

1 What Is Lean Science and Why Should I Care?

The term “Lean” was originally coined in the 1980s to denote a set of management practices and principles used very successfully at Toyota in the preceding decades. Like most management practices, Lean deals with both technical and people-oriented areas. More recently, Alan Shalloway described Lean as consisting of three bodies of knowledge (Shalloway 2010):

- Lean Science: Batch size, WIP limits, Pull management, Cadence, Flow
- Lean Management: Visual Controls, Leadership, Education
- Lean Knowledge Stewardship: A3, 5 Whys, Continuous Improvement, Kaizens

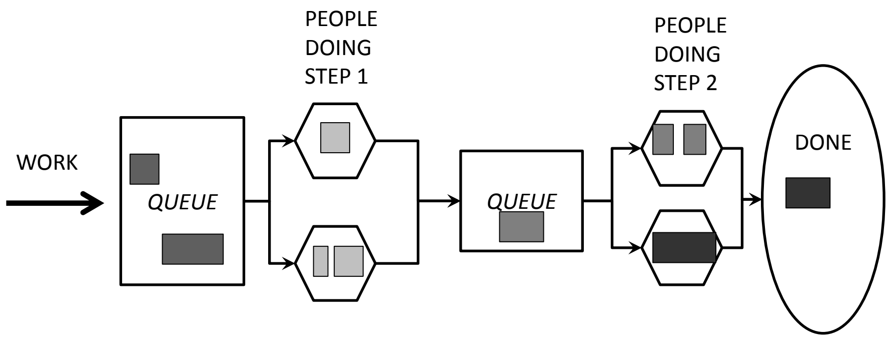
In this paper, we’ll talk mostly about Lean Science, sometimes known as Second-Generation Lean (Reinertsen 2009). The intent of Lean Science is to optimize an organization so it will produce more output over time. Lean Science is grounded in a body of mathematics known as *queuing theory*, which predicts how work flowing through an organization will behave. Queuing theory is used to design phone systems, Internet networking, traffic control systems, and other types of systems, so we know a lot about it and we know that it works.

Luckily, it’s not necessary to learn a lot of math in order to benefit from Lean Science and queuing theory. You do need to understand the basic model involved – how to see your work flowing through your organization. I’ll then describe the steps I use to get the work under control, using principles derived from queuing theory. We’ll look at how to apply those steps in a software test lab to achieve a more effective, less overloaded organization.

1 Lean Science – a Systems View

The key to Lean Science is the *systems view* - looking at the organization as if it were a system or machine for turning work requests into finished, saleable work. Lean Science is all about understanding how work flows through your system.

A systems view of a typical organization is shown below. The little boxes represent chunks of work flowing through the system. These chunks are known as *batches*. They are different sizes and arrive somewhat randomly.



At any given time, a chunk of work is either being worked on (hexagons), or waiting in a *queue* for somebody to work on it (rectangles).

The goal of Lean Science is to make the batches reach “Done” faster, without increasing the number of people doing the activities. What other parameters can be changed, and what effect will changes have? The consequence of a change is often not intuitively obvious. We don’t seem to be wired very well to reason about systems like this. However, queuing theory identifies the parameters for us and provides ways to predict what will happen if a parameter is changed.

2 Visibility First, Then Improvement

The first step in applying Lean Science is to make our work visible. We'll change the way we look at, talk about, and measure our work. This will enable us to clearly see how much value is created each week or month. The basic steps are these:

- Split our work into small, clearly defined *batches,* each of which delivers value.
- Map our system by identifying the activities and the wait states (*queues*).
- Make the progress of each batch visible.

Just making the work visible often creates improvement by making duplication, unnecessary work, and poorly defined handoffs obvious.

Once we can see our work, and we can see how much value is (or isn't) being produced, we use a few concepts from queuing theory to look for potential improvements. Typical improvements include (not in any particular order).

- Manage the batches on a cadence.
- Limit the number of batches under development at any given time.
- Reduce batch size.

This is a general approach that works for all types of "knowledge work" - software development, mechanical engineering, pharmaceutical research, etc. However, in this paper, we'll assume that the organization is a software test lab.

3 Make the Work Visible

3.1 Split the Work into Batches Which Deliver Value

The work must first be split into discrete batches. In manufacturing, this is easy to do since the work consists of physical items flowing through a manufacturing line. In software testing, we've got to decide what will constitute a batch or chunk of work for our particular lab.

Applying Lean Science will maximize the output of batches, regardless of what those batches consist of. If we simply list tasks as our batches, we will maximize the output of tasks. This is fine if your lab is recompensed by the hour. If your value is assessed any other way (which is commonly the case), each of your batches needs to deliver some value ***as perceived by your customers***. Otherwise, you'll simply maximize the amount of effort you put in, not the amount of value you produce.

Here are some typical batches which deliver value:

- Writing or finding the appropriate tests to adequately cover a newly delivered feature, running the tests, and reporting the defects in a clear, reproducible manner.
- Designing and running in-depth performance tests and reporting what performs adequately and what doesn't.
- Writing and running localization tests and reporting the defects.

- Running a regression test and delivering an assessment of readiness to release or a prediction of the likelihood of unfound defects.
- Preparing a status report on progress to date, trouble spots, and perhaps a prediction of when the software will be ready to release.

All these batches are delivering information which is valued by the development team. The developers value well-written defect reports, because those help them quickly identify the root cause of the defect. The managers value lists and totals of unfixed defects or tests still to be done, because those help them assess how much work remains to be done. Advocates for the end-user appreciate defect characterization and severity ratings, because they want to know what the end-user will see if a particular defect is not fixed. Management also values any good prediction you can make regarding how many ***unfound*** defects are still in the product. The point of the batch is not to run tests *per se*, but to acquire and deliver information.

Most test labs also have batches whose value is to reduce the lab's overhead, rather than directly provide anything to outside customers. The customer in this case is the test lab's own management. Some examples:

- Design and write a set of tests for our reusable test library.
- Automate a set of tests.

And then there's always the stuff that we just have to do:

- Move equipment due to a space reassignment.
- Hire a new employee.

So, there'll be several different types of batches, and the batches will be different sizes. That's all fine. The important points are these:

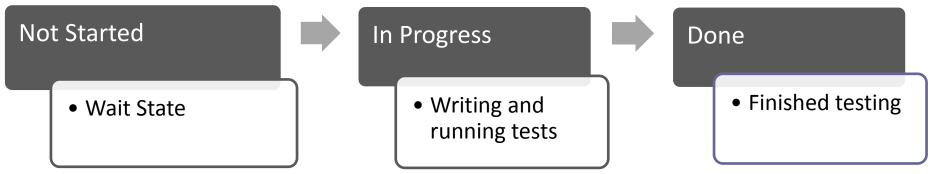
- All of your work is captured in batches.
- The value of each batch is clearly stated.
- Each batch's exit criteria are clearly defined, so we can tell when each batch is done.

3.2 Map Your System

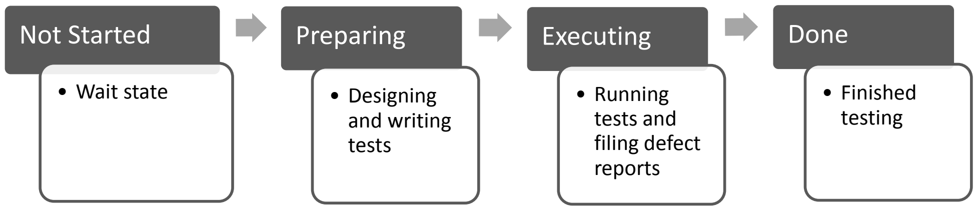
3.2.1 A Basic Map

Now you've got a list of the batches of work which your organization performs. Let's think about what happens to each batch as it moves through your organization. Who decides when a batch will be started? What are the major steps which must be performed? Will the batch be handed off between individuals or teams specializing in different areas?

Since Lean Science is meant to optimize your lab as whole, the same map will be used for all the types of work you do. This means it needs to be quite general. The simplest map has just three states, as shown to the right.



The three-state map is often sufficient for smaller teams. It might also work for a larger, more complex organization, or the larger organization may want another state or two to show the handoffs between specialized teams, as shown below.


Not Started
Wait state
Preparing
Designing and writing tests
Executing
Running tests and filing defect reports
Done
Finished testing

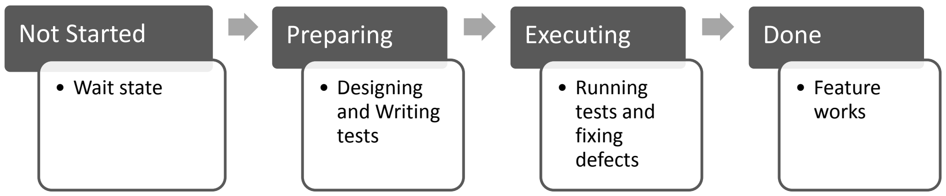
Some people call this type of map a process map and others call it a value-stream map. However, this map usually is less detailed than either a formal value-stream map or a formal process map. It's better to start simple, and add detail only as you need it. It's not necessary to capture the total reality of the process – you just need enough to give you useful insights.

3.2.2 Thoughts on System Optimization

You may notice that all this is going to optimize *the test lab*, not the overall organization. Will this help the overall organization make more money or be more effective? If the organization sells testing services, it will. If the overall organization sells software, or sells something else and is using the software to assist in the something else, then making the test lab more productive may not help the bottom line. It can even make things worse! This depends on whether testing is or is not a bottleneck in the production of the software.

Most organizations will argue that testing is a bottleneck, at least some of the time. In reality, changes in the development process might help more than changes in the test lab, but the test lab manager normally doesn't make decisions at that level. The bottom line: use Lean Science to optimize the area of the organization which you control, but keep in mind that you are only a piece of the whole. If you get the opportunity to participate in optimizing a wider area, take it!

One way to make sure that the whole system is being optimized is to focus your batches on feature completion, not on testing. Change the definition of "done" from "finished running the tests" to "this feature works", as shown in this process map:

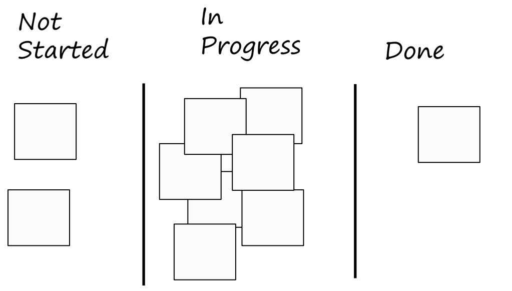


In a completely integrated agile development system, this is already the way things work. In a waterfall situation, or where the test lab is integrating the work of several unsynchronized agile teams, this method can be applied. (Iberle 2010) As we'll see later, this approach also allows the test lab to predict the end date of the project fairly accurately.

3.3 Make Each Batch and Its Progress Visible

Once all your work is expressed in batches, it must be made visible in a concrete way.

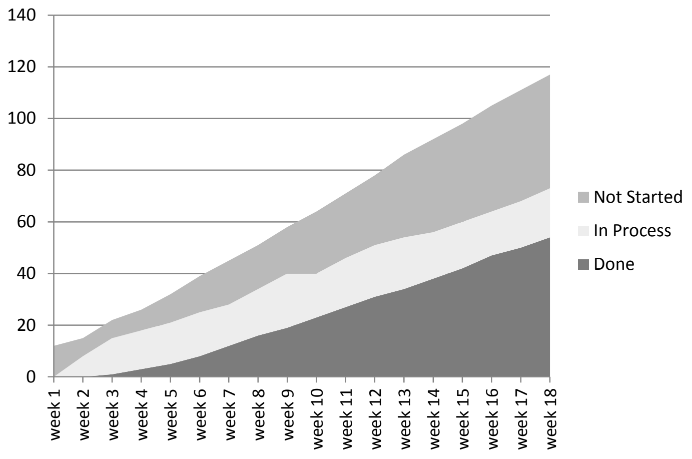
The simplest way to make work visible is to create a list of the batches. I've seen this cause improvement all by itself, particularly in large multi-site organizations. Once all the work is listed in one place, it's easy to see duplications and work that was supposed to have been discontinued. Since the batches are stated in terms of value, it's also easier to see why the work is being done.



The next level up is the Visual Planning Board, which tracks not only the existence of the work but also its current status. Your process map is depicted as columns on a wall or whiteboard, and each batch is represented by a sticky note in the appropriate column. It's much easier to grasp the "big picture" with this visual format, which makes day-to-day problems more visible.

The third tool, and the most powerful, is the *Cumulative Flow Diagram* or CFD. The Visual Planning Board is a snapshot in time. The CFD tracks progress trends over time.

Each week, count how many batches are in each state. Graph the totals as a stacked area chart, with "Done" on the bottom and "Not Started" on the top, as shown to the right.



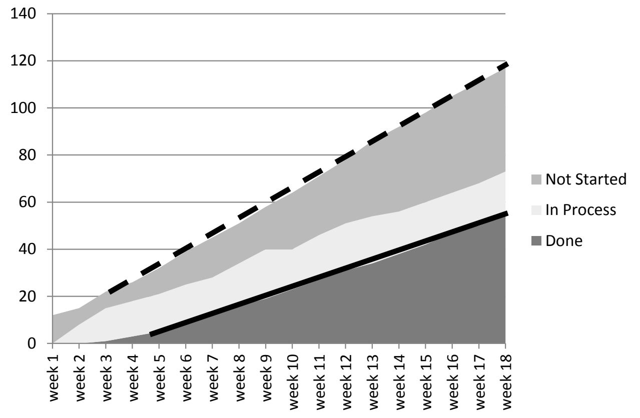
For all you testers: Have you used a chart like this to track defect trends over time? If so, you already know quite a bit about reading this chart.

For all you agilists – yes, this is a "burn-up" chart. Notice that it's burning up *batches*, not tasks. This keeps us focused on improving throughput of batches, not throughput of effort.

3.4 What Can the Cumulative Flow Diagram Tell Me?

Let's see what we can learn from our CFD.

3.4.1 Are We Getting Behind?



The solid line on the CFD to the right shows how many batches are delivered each week. This is known as the *throughput*.

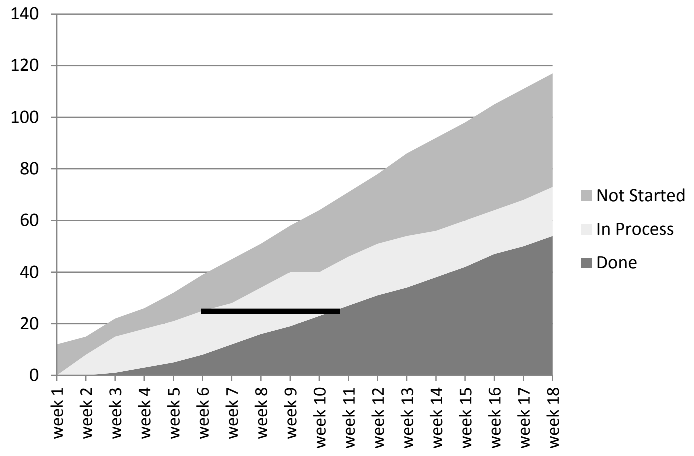
The dashed line shows how many batches are requested each week. This is the *demand*.

If work is requested faster than it can be done, the work is going to pile up. The CFD makes the problem visible, in a way that is compelling in management meetings.

3.4.2 How long does it take for the average item to finish?

A team can have good throughput but still be criticized for the length of time it takes to address a newly submitted item. The time from the start of active work to completion is known as the *cycle time*.

The average cycle time can be read right off the CFD by drawing a horizontal line across the “In Process” area. In this example, the average item spends four and a half weeks “In Process”. The average cycle time can change over time. We’ll discuss that in a later section.



3.4.3 When will we be done?

If your batches are designed around features or capabilities of the product, your CFD might be able to predict when the product will be ready to release.

You *can* predict the future if these criteria are met:

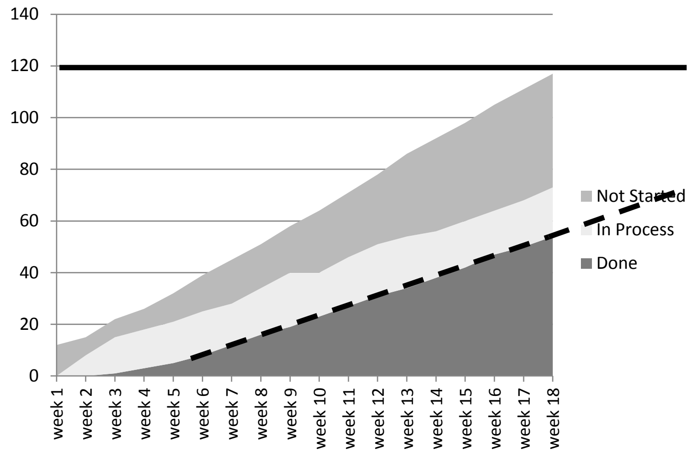
- Each batch represents a useful feature and/or capability.
- The batch is not considered “done” until the feature or capability is *passing* the tests.
- Only the batches representing testing of a given product or release are shown. The CFD for this product doesn’t include batches for ongoing lab maintenance, testing of other projects, etc.
- There is a list of the probable total features and/or capabilities that need to be finished before the product can release. Your test plan probably contains such a list.
- The batches are small enough that the total project contains “a lot”. I like to see at least fifty batches.

In this CFD,

- The dashed throughput line shows how fast the batches are getting done.
- The solid line represents the approximate number of batches which are planned for this project.

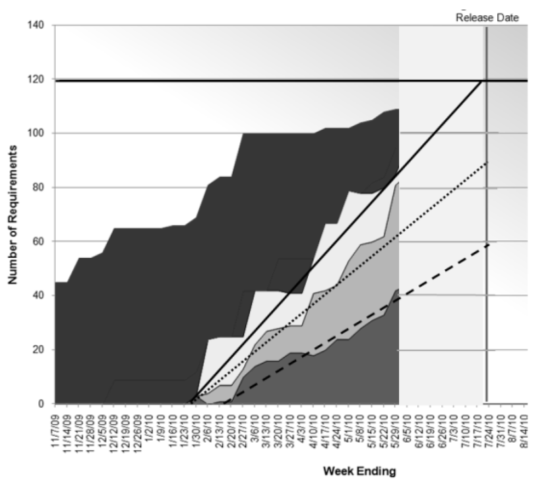
The approximate end date of the project can be estimated by extending the throughput line until it meets the planned-batches line. Agile teams often do this type of prediction using their burn-up or burn-down charts.

Of course, real data is much messier than this “toy” example.



This CFD contains data from a real project (Iberle 2010).

- The solid throughput line represents features entering system test.
- The middle dotted line represents features which have been tested but have not yet passed the tests. They are waiting for a fix, or being fixed, or being re-tested.
- The dashed line represents features completed.



All three throughput lines are fitted to the actual data. It's possible to use least-squares regression but most people simply draw a line that looks like it fits, as was done in this case. The distance between the line and the actual data points on either side of it gives you an idea of the accuracy of the prediction.

The prediction is more accurate if many batches are involved, just like the total of many coin-flips will be very close to 50% heads and 50% tails. Very small teams won't get much help from a CFD because they just don't have enough data points to form a pattern. It also helps if the average size of the batches doesn't radically change between one time point and the next.

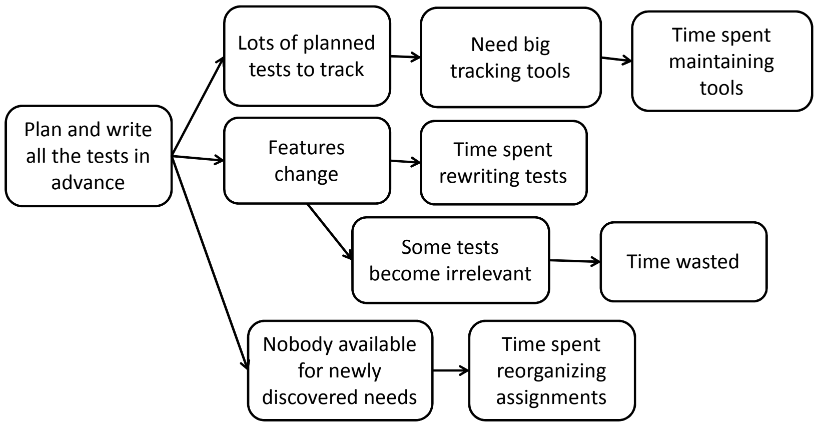
The CFD provides an unparalleled view of the long-term performance of an organization, regardless of the exact nature of the work performed from day to day. We'll see some more ways to use the CFD as we proceed.

4 Improvements in Your Test Lab

Now that you've got your tools in place, you can make improvements in the throughput of your test lab. As our batches move through our system, we want to reduce waste (unneeded work) and delay (batches waiting instead of moving along).

Software development and software testing both consist of **variably sized** chunks of work arriving at a **variable and unpredictable** rate. Queuing theory tells us that the most common and nefarious sources of waste and delay in such a system are

- excessively large batch size
- too much work-in-process


Plan and write all the tests in advance
Lots of planned tests to track
Need big tracking tools
Time spent maintaining tools
Features change
Time spent rewriting tests
Some tests become irrelevant
Time wasted
Nobody available for newly discovered needs
Time spent reorganizing assignments

In his 2009 book *Principles of Product Development Flow*, Reinertsen lists ten major negative effects of excessive batch size. We're often so used to the effects of large batches that we don't even see them. For instance, in some organizations the standard process calls for all the tests to be written before the first build of the product arrives for testing. On the surface, this sounds like good practice. But when you look at it carefully, there is a lot of waste, as shown above.

The other major, and perhaps more insidious, source of waste is *Work-in-Process* or WIP. The waste here is caused by the cost of task-switching. A human being cannot switch from one task or topic to another for free. The simplest switches take a few seconds. In a switch between complex tasks, the worker may take as much as ten minutes to come fully up to speed on the second task. The worker can get the most work done, with the least stress, when he or she works on only two or three different things in a given day.

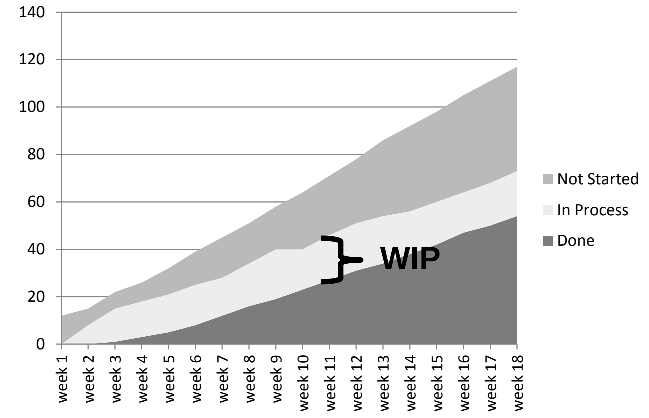
In addition, when a number of batches are in progress at the same time, the organization has to keep track of them. This means creating and maintaining lists, tracking systems, status reports, and so forth. Most organizations are investing a lot of time and money in keeping track of half-finished work. If you limit the Work-in-Process – finish items before starting new ones – both the task-switching time and the tracking work can be cut dramatically.

I'm going to talk about WIP first and then batch size, because simply defining the batches as we did in the earlier sections often corrects the batch size enough to get started. You may have different results.

By the way, if you're familiar with Lean, you may be wondering why I'm not talking about the "seven wastes" or "one-piece flow". These rules are derived from applying queuing theory to a system of **consistently sized** chunks of work arriving at a **steady and predictable** rate, such as you find in manufacturing or health care or office management. Systems of variably sized work behave somewhat differently, so the rules derived from queuing theory for variable systems are not exact analogues of the rules for consistently sized systems. In addition, product development is meant to discover new knowledge, so variation is sometimes valuable rather than always being detrimental.

4.1 Limit the Number of Batches in Progress

The current work-in-process or WIP can be read off the CFD as shown to the right. This shows the WIP of the whole department. Ideally, no individual in the department is working on more than two or three batches at the same time. The ideal WIP limit for the department thus is usually twice or three times the number of people.

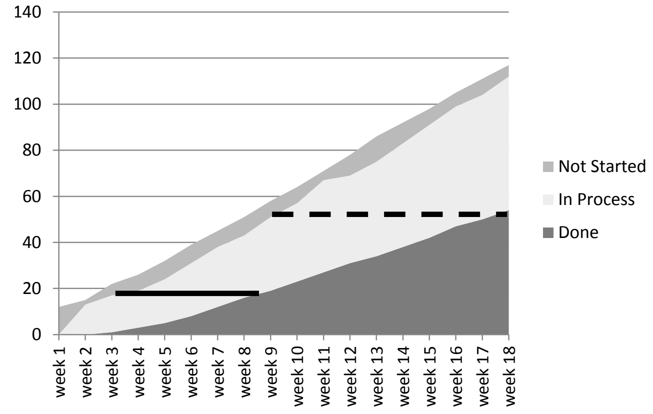


This seems counter-intuitive to many people. If there are more requests coming in, we should start more items. Everyone will see that we're working hard and using all our people to full capacity, right?

Wrong!

Let's see why…

Here's a CFD for an organization which keeps taking on new batches before the older ones are finished. Its WIP is gradually increasing.



The two lines show the average time elapsed between starting an item and finishing an item. In week 3, the average elapsed time is about 5 weeks. By week 9, the average elapsed time has grown to 9 weeks. As the WIP grows, the average elapsed time grows, which means that responsiveness drops.

In addition, the overhead caused by the increased WIP makes the throughput decrease. Now you're taking longer to get less done. This doesn't impress your departments' customers.

This effect is particularly troublesome for batches representing internal improvement initiatives, which are usually getting only your "left-over" time in the first place. In a large lab, it's frighteningly easy to start so many worthwhile initiatives that almost nothing gets done.

It may seem impossible at first to limit your Work-in-Process. The key is to "Stop Starting, Start Finishing". You've got to stop starting new batches until some of the old batches are finished. Start with the work that is under your direct control, such as improvement initiatives. Stop starting new ones until you've finished some of the older ones. You will gradually be able to expand this to include testing batches. The discipline is easier if batch size is kept small. When batches are small, more batches will finish each week, providing more opportunities to start something new or change priorities.

4.2 Reduce Batch Size

Now that you can see and control your WIP, it's time to talk again about batch size.

What is the right size for a batch? The ideal batch size is a trade-off between the positive aspects of getting value more frequently, and the negative aspects of managing lots of small batches. When the management costs get too high, your batches are too small. Don Reinertsen gives much more detailed explanation, with the math, in *The Principles of Product Development Flow* (Reinertsen 2009 Ch. 5).

You can approximate the math by listing both the costs and the benefits of the batch size, so you can compare the effects of changing the batch size.

Typical benefits of small batches:

- Earlier delivery of valuable results to your customers.
- Faster feedback on how long this batch really took. (As we observed earlier, cumulative information on actual elapsed time improves schedule predictions.)
- More frequent opportunities to respond to newly discovered needs without leaving half-finished work behind.
- Reduced risk. You'll find out sooner if your test equipment isn't adequate, or the network keeps failing, or the build system isn't reliable.
- Earlier opportunities to realize that further testing on this build will just waste time and money.

Typical costs of small batches:

- Keeping track of all the batches – maintenance of tools, routine meetings to review the batches.
- The effort needed to define each batch.

One rule of thumb is to make the batch sizes small enough to change people's assignments reasonably often. For instance, if you'd like to change people's assignments every two weeks, then the average batch size must be around two weeks or less. Anything larger will cause your system to "clog up" and your responsiveness will go down. Another rule of thumb is to make the batches correspond to specific questions to be answered. For instance, a week's delivery from a development team of ten probably corresponds to a couple of test batches, not just one.

It takes some practice to get good at defining batches that are both useful and small, particularly for improvement initiatives. At first, people have trouble imagining that anything useful can be delivered without doing the entire project. Agile development discussions of “splitting user stories” can be helpful, particularly when you’re looking at batches which are improvement projects. Try using the Extreme Programming mantra “What is the simplest thing that could possibly work?” (Beck 2000).

4.3 Manage the Batches on a Cadence

A cadence is a regular, predictable rhythm within a process. For instance, a particular meeting happens on the same day of every week, or your website is refreshed on the last day of the month every month.

A cadence saves time by reducing waste. When an activity happens at a predictable time, people plan their work to take advantage of the cadence. A cadence also saves time by reducing overhead. Instead of coordinating multiple people’s availability and finding a meeting room each time the meeting is needed, the calendar coordination and meeting room reservation is done just once.

Batches are managed on a cadence by making decisions about which batches to do, or not do, on evenly spaced dates. Ideally these dates are quite close together – every week, every two weeks, or every four weeks. This reduces the need to discuss priorities repeatedly and maintain lists in priority order. Instead, every two weeks, the group decides which are the most important things to do *in the next two weeks*. Since only a limited number of things can be done in the next two weeks, the team focuses on just those and decisions can be made quickly. Little or no time is spent debating the relative priorities of items in the far future.

Of course, if you’re working within an agile method already, there are at least two cadences already in place – the sprint cadence and the standup meeting cadence. If you’re not, establishing a regular cadence to review the planned testing (preferably with the developers!) will probably cut your overhead. Your internal improvement projects can be managed on a similar cadence, although probably in a different meeting. David Anderson’s book *Kanban: Successful Evolutionary Change for Your Technology Business* (Anderson 2010) describes Kanban in fairly general terms, providing a structure which is usable by groups who aren’t doing software development.

Think about other ways a cadence might benefit you. For instance, if your senior people review the work of more junior people, regularly scheduled “office hours” are likely to be more efficient than interrupting every time someone is ready for a review. Any time someone has a part-time activity working with or helping someone else, applying a cadence might help.

5 Case Studies

Most of the methods in this paper are based on Don Reinertsen’s work in *The Principles of Product Development Flow: Second-Generation Lean Product Development* (Reinertsen 2009). This book is a general treatment of the principles and how they can be applied in a variety of fields, with a few examples involving software testing. The usefulness of the methods is demonstrated by (among other examples) the success of agile software development, although I suspect that many agile practitioners don’t realize that a large part of their success is due to controlling the WIP and batch size. In his book *Agile Software Requirements*, Dean Leffingwell describes how the theory behind Second-Generation Lean applies to software development in general. (Leffingwell 2011).

However, there is very little published on software testing per se. My personal experience with this method is captured in three publications:

- “Introducing Fast Flexible Flow at Hewlett-Packard”. (Iberle 2012). This paper describes how these methods were initially used to get better control of the work flowing through a large system test lab.

- “Kanban: What is it and why should I care?” (Reese 2011). Landon Reese and Kathy Iberle describe the next step taken by the same group, which was implementation of a Kanban system to handle tooling work. This system was later expanded to manage test writing and test execution requests.

- “Lean System Integration at Hewlett-Packard” (Iberle 2010). This paper describes the first year or two of managing the integration of a large system, including the prediction of project end dates. More progress was made since the publication of that paper, but has not been published.

Matt Heusser shares some thoughts on a Lean approach to software testing in his blog “Applying Lean Concepts to Software Testing” (Heusser 2010).

6 Bottlenecks, Value Stream Maps, and Other Topics

This paper has focused on software testing as an activity that consists of **variably sized** chunks of work arriving at a **variable and unpredictable** rate. However, most test labs do have activities where the work is both **consistently sized** and arrives at **a predictable rate**. These tend to be repetitive activities, such as installation of operating systems, build-and-smoke-test procedures, or regression testing. In this case, the methods used in Lean Manufacturing can be very helpful.

The Lean Manufacturing approach typically starts by creating a detailed value-stream map, and then using the map to look for bottlenecks and wasted effort. Descriptions of this method are readily available from many sources, such as the Lean Enterprise Institute website. (Lean Enterprise Institute 2013). If the you find that the waste is not readily visible or the bottleneck moves around, I suggest trying the approach in this paper instead.

7 Conclusion

Applying Lean Science to a software test lab is not a trivial activity, but it can be very rewarding. The first, and often the most difficult part, is to re-define the work into small, discrete batches which have clear end points and independently deliver value. A simple system map is also necessary. Simply defining the work in batches and making the batches visible often creates improvement in several ways:

- The increased visibility can expose unnecessary work – accidental duplication, failure to communicate a cancellation, etc.

- Defining the value of the information to be delivered by each batch can reveal testing that is scheduled too early, excessively thorough, or simply not needed.

- Establishing the criteria for “done” can resolve long-standing mismatches between subteams.

Once your work can be viewed as batches flowing through your system, you have a large number of tools at your disposal to manage that work more efficiently. The Visual Planning Board provides a quick and easy way to communicate status. The Cumulative Flow Diagram provides a view of the long-term performance of longer projects and of the lab itself.

Once these tools are available, the organization can improve its performance by going after the most likely candidates for waste in a variable system – excessive WIP and excessive batch size. Other Lean methods can also be applied, such as cadence.

Good luck with your journey into Lean Science!

8 References

Anderson, David J. 2010. *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim, WA: Blue Hole Press.

Beck, Kent. 2000. *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley.

Heusser, Matt; “Applying Lean Concepts to Software Testing” (posted November 2010). http://searchsoftwarequality.techtarget.com/tip/Applying-lean-concepts-to-software-testing?asrc=EM_NLN_12859187&utm_medium=EM&utm_source=NLN&utm_campaign=20101116_Today's%20News:%20Applying%20lean%20concepts%20to%20software%20testing_mewebb&track=NL-498&ad=798129 (accessed July 10th, 2013).

Iberle, Kathleen A. 2010. “Lean System Integration at Hewlett-Packard”. Proceedings of the Pacific Northwest Software Quality Conference 2010.

Iberle, Kathleen A. 2012. “Introducing Fast Flexible Flow at Hewlett-Packard”. Lean Product and Process Development Exchange, 2012.

Iberle, Kathy. 2013. “Finding the Best Frequency for a Recurring Activity”. http://www.kiberle.com/2013KB/CadenceMath.pdf (accessed July 10th, 2013).

Leffingwell, Dean. 2011. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Boston: Addison-Wesley Professional.

Lean Enterprise Institute. http://www.lean.org/WhatsLean/Principles.cfm. (accessed August 09, 2013).

Reese, Landon. Iberle, Kathleen A. 2011. “Kanban: What is it and why should I care?”. Proceedings of the Pacific Northwest Software Quality Conference.

Reinertsen, Donald G. 2009. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Redondo Beach, CA: Celeritas Publishing.

Shalloway, Alan. Beaver, Guy. Trott, James R. 2010. *Lean-Agile Software Development: Achieving Enterprise Agility*. Boston, MA: Pearson Education. pp. 217-218.

Kathy Iberle’s publications can be found at http://www.kiberle.com/articles.html