Dan Slimmon

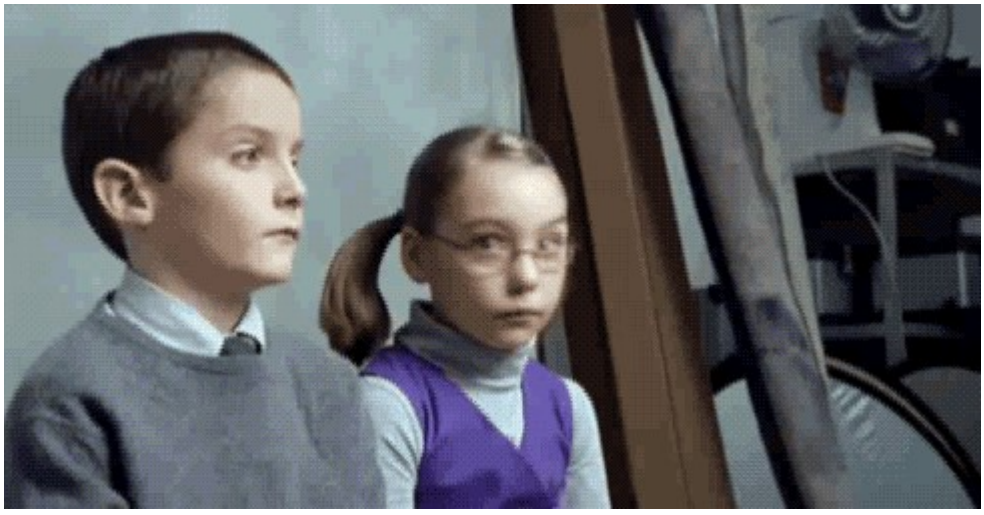
SRE at Hashicorp

When efficiency hurts more than it helps

ON 2015/07/092015/07/14 / BY DAN SLIMMON / IN WORKFLOW

When we imagine how to use a resource effectively – be that resource a development team, a CPU core, or a port-a-potty – our thoughts usually turn to *efficiency*. Ideally, the resource gets used at 100% of its capacity: we have enough capacity to serve our needs without generating queues, but not so much that we're wasting money on idle resources. In practice there are spikes and lulls in traffic, so we should provision enough capacity to handle those spikes when they arrive, but we should always try to minimize the amount of capacity that's sitting idle.

Except what I just said is bullshit.



In the early chapters of Donald G. Reinertsen's brain-curdlingly rich Principles of Product Development Flow (http://www.amazon.com/The-Principles-Product-Development-Flow/dp/1935401009), I learned a very important and counterintuitive lesson about queueing theory that puts the lie to this naïve aspiration to efficiency-above-all-else. I want to share it with you, because once you understand it you will see the consequences everywhere.

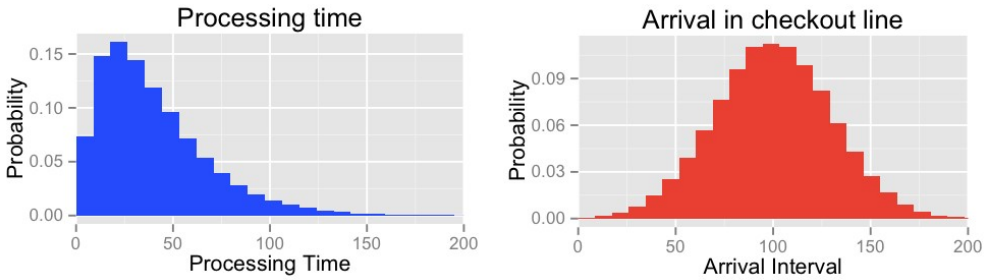
Queueing theory?

Queueing theory is an unreasonably effective (https://www.dartmouth.edu/~matc/MathDrama/reading/Wigner.html) discipline that deals with systems in which **tasks** take time to get **processed**, and if there are no processors available then a task has to wait its turn in a **queue**. Sound familiar? That's because queueing theory can be used to study basically anything.

In its easiest-to-consume form, queueing theory tells us about average quantities in the steady state of a queueing system. Suppose you're managing a small supermarket with 3 checkout lines. Customers take different, unpredictable amounts of time to finish their shopping. So they arrive at the checkout line at different intervals. We call the interval between two customers reaching the checkout line the **arrival interval**.

And customers also take different, unpredictable amounts of time to get checked out. The time it takes from when the cashier scans a customer's first item to when they finish checking that customer out is called the **processing time**.

Each of these quantities has some variability in it and can't be predicted in advance for a particular customer. But you *can* empirically determine the probability distribution of these quantities:



(https://danslimmon.files.wordpress.com/2015/07/distributions1.png)

Given just the information we've stated so far, queueing theory can answer a lot of questions about your supermarket. Questions like:

- How long on average will a customer have to wait to check out?
- What proportion of customers will arrive at the checkout counter without having to wait in line?
- Can you get away with pulling an employee off one of the registers to go stock shelves? And if you do that, how will you know when you need to re-staff that register?

These sorts of questions are super important in all sorts of systems, and queueing theory provides a shockingly generalizable framework for answering them. Here's an important theme that shows up in a huge variety of queueing systems:

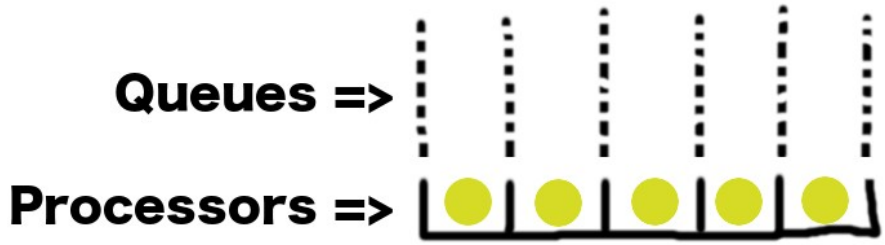
The closer you get to full capacity utilization, the longer your queues get. If you're using 100% of capacity all time, your queues grow to infinity.

This is counterintuitive but absolutely true, so let's think through it.

What happens when you have no idle capacity

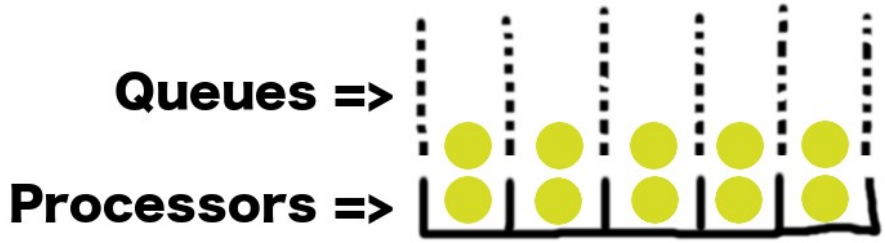
What the hell? Isn't using capacity efficiently how you're supposed to get *rid* of queues? Well yes, but it doesn't work if you do it all the time. You need some buffer capacity.

Let's think about a generic queueing system with 5 processors. This system's manager is all about efficiency, so the system operates at 100% capacity all the time. No idle time. That's ideal, right?



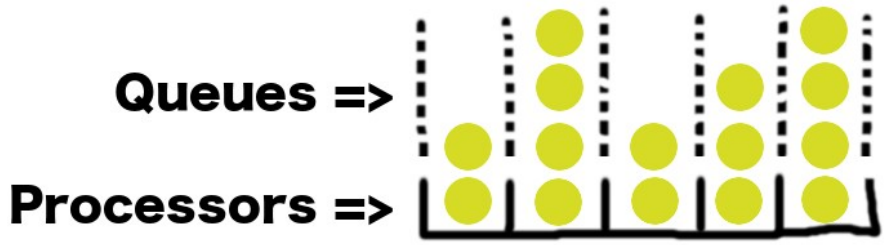
(https://danslimmon.files.wordpress.com/2015/07/fullcap-0.png)

Sure, okay, now what happens when a task gets completed? If we want to make sure we're always operating at 100% capacity, then there needs to be a task waiting behind that one. Otherwise we'd end up with an idle processor. So our queueing system must look more like this:



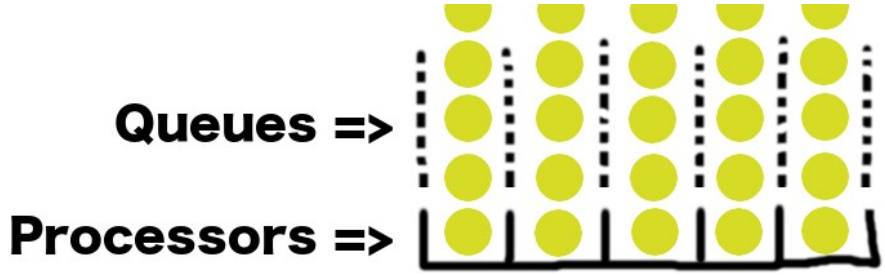
(https://danslimmon.files.wordpress.com/2015/07/fullcap-1.png)

In order to operate at 100% capacity all the time, we need to have at least as many tasks queued as there are processors. But wait! That means that when another new task arrives, it has to get in line behind those other tasks in the queue! Here's what our system might look like a little while later:



(https://danslimmon.files.wordpress.com/2015/07/fullcap-2.png)

Some queues may be longer than others, but no queue is ever empty. This forces the total number of items in the queue to grow without limit. Eventually our system will look like this:



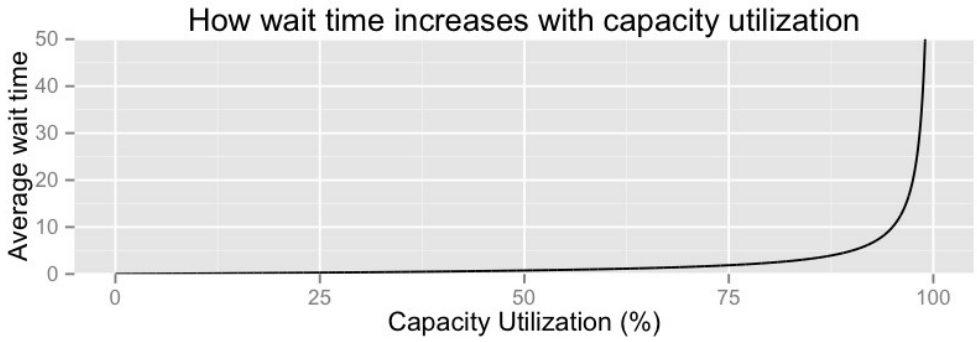
(https://danslimmon.files.wordpress.com/2015/07/fullcap-3.png)

If you don't quite believe it, I don't blame you. Go back through the logic and convince yourself. It took me a while to absorb the idea too.

What this means for teams

You can think of a team as a queueing system. Tasks arrive in your queue at random intervals, and they take unpredictable amounts of time to complete. Each member of the team is a processor, and when everybody's working as hard as they can, the system is at 100% capacity.

That's what a Taylorist (https://en.wikipedia.org/wiki/Scientific_management) manager would want: everybody working as hard as they can, all the time, with no waste of capacity. But as we've seen, in any system with variability, that's an unachievable goal. The closer you get to full capacity utilization, the faster your queues grow. The longer your queues are, the longer the average task waits in the queue before getting done. It gets bad real fast:



(https://danslimmon.files.wordpress.com/2015/07/quartz_3____.png)

So there are very serious costs to pushing your capacity too hard for too long:

- Your queues get longer, which itself is demotivating. People are less effective when they don't feel that their work is making a difference (see The Progress Principle (http://www.amazon.com/Progress-Principle-Ignite-Engagement-Creativity-ebook/dp/B0054KBLBI/ref=tmm_kin_swatch_0?_encoding=UTF8&sr=&qid=))
- The average wait time between a task arriving and a getting done rises linearly with queue length (https://en.wikipedia.org/wiki/Little's_law). With long wait times, you hemorrhage value: you commit time and energy to ideas that might not be relevant anymore by the time you get around to them (again: read the crap out of Principles of Product Development Flow (http://www.amazon.com/The-Principles-Product-Development-Flow/dp/1935401009))
- Since you're already operating at or near full capacity, you can't even deploy extra capacity to knock those queues down: it becomes basically impossible to ever get rid of them.
- The increased wait time in your ticket queue creates long feedback times, nullifying the benefit of agile techniques.

Efficiency isn't the holy grail

Any queueing system operating at full capacity is gonna build up giant queues. That includes your team. What should you do about it?

Just by being aware that this relationship exists, you can gain a lot of intuition about team dynamics. What I'm taking away from it is this: *There's a tradeoff between how fast your team gets planned work done and how long it takes your team to get around to tasks.* This changes the way I think about side projects, and makes me want to find the sweet spot. Let me know what you take away from it.

PROBABILITY QUEUEING-THEORY WORKFLOW

5 thoughts on “When efficiency hurts more than it helps”

1. **Dieter Plaetinck**
   Hey Dan!
   I should preface this by saying I’m not a queueing theory expert.. but it seems this logic assumes that the average job arrival rate is higher than the processing rate. In that case I can see how the queues grow infinitely.
   But think about this: let’s say you’re processing jobs and have a few jobs lined up for each processor. If you can control the input rate to equal, on average, the processing rate, then the queues shouldn’t grow, right?
   This approach is basically JIT (just in time) delivery (https://en.wikipedia.org/wiki/Just-in-Time_Manufacturing)
   It seems the catch is that the system needs to support arbitrary sequences of arrival rates, including prolonged timeframes of very low arrivals – which can still satisfy the given distribution if we have enough time to make up for a temporary “imbalance” – without dropping below 100% utilization. And maybe that’s the underlying reason why queues have to keep growing, to support any possible temporal “distribution imbalance”.
   If we were to restate the target as “100% utilization during at least % of a number minutes” or “100% utilization as long as any window of x seconds/minutes long meets a certain distribution, or as any such window has an average arrival rate of y per second/minute” then we should have an upper bound on how large queues have to be, given a known processing rate.

   2015/07/12 AT 20:44 ↪ REPLY
   Dan Slimmon
   You would absolutely LOVE Principles of Product Development Flow. It the traditional minimize-variability approach and points out all the mathematical subtleties that differentiate product development from manufacturing.

   In the formulation of queueing theory I’m using here, we talk about the overall distribution and assume the distribution remains constant, so there’s no way to get a temporal “distribution imbalance.” And you’re right that for steady-state 100% utilization to occur the arrival rate must be greater than or equal to the processing rate. However, in coding & ops, unavoidable sources of variability join forces with high costs of delay to make it so we can’t even operate *close* to 100% utilization without severe penalties.

   At 95% utilization for example, we do have some times (5%) when there’s no work in the queue. This is very close to the

naïve ideal of 100% utilization. But as seen on the “Average wait time vs. Capacity Utilization” chart, *average* queue length (linearly proportional to average wait time by Little’s Law) blow up to crazy numbers at 99% utilization.

This causes big problems for the organization because long queues gum up the JIT feedback cycle (new work takes forever to get through the pipe), because of the overhead of queue management, and also because there are strong economic incentives to evaluate opportunities sooner rather than later. Plus, long queues are just downright demotivating.

So what Reinertsen recommends in his book is to think about the tradeoff between capacity utilization and queue length, and try to find the sweet spot.

2015/07/14 AT 11:38 ↳ REPLY

2. Pingback: Minding Your Pees And Queues | Empiricism Ops

3. Pingback: The most important thing to understand about queues – Dan Slimmon

4. Pingback: [Back from NewCrafts] Des flux et des files d’attente | Le blog technique de Younited Credit