Agile and congestion

Author: Han Schaminée

Contributors: None yet

Version 1.0

Introduction

The concept of congestion in software development is in my view still very little understood. A lot of research needs to be done and I invite universities to put this on their research agenda. But it is already worthwhile to ask attention for this serious problem and trigger a discussion via my discussion papers. In manufacturing and in telecommunications, we know for many years efficiency decreases if you plan for all resources to be fully utilised. In software engineering, pressure seems to be so high, we have no time to think about better ways and rather continue to struggle. Like no time to switch gear. It requires strong management to break through this. This document describes some of the problems I have experienced.

Executive summary

Optimizing utilisation of scarce resources may have a negative effect on total throughput because there is often intrinsic uncertainty in software development. High utilisation of development resources can cause congestion, like in traffic. You can detect your organisation is congested when throughput shows a structural decline, but also things like reduced quality, non-transparent reporting or can-not-do mentality are clear indicators. The best way to avoid congestion is to plan for spare capacity, have a strong focus on cycle time and create transparency on progress, issues and risks. The root cause for congestion is often a manager who does not understand the basics of Agile development and needs help to apply its principles. Repairing a situation of congestion is even more difficult and requires more drastic measures. But in the end, it is worthwhile doing.

What is congestion in software development?

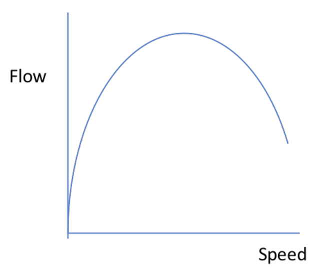
No buffers on the critical path

In his book “Critical Chain”, E.M. Goldratt applies the theory of constraints on project management. He stresses organising around optimization of the bottlenecks. He advises to have no buffers on the critical path, only feeding buffers for those paths that contribute to the critical path. He explains there is a big productivity risk with buffers on the critical path, as buffers tend to fill (and this seems to be very true in software development).

In his book “The principles of product management”, Donald G. Reinertsen explains how this model does not fully apply when uncertainty comes into play. You need a buffer at the end of the critical path, just before the commitment, to deal with the intrinsic uncertainty of product development. The lack of such a buffer leads to congestion.

Too much traffic on a highway creates congestion

Congestion, according to Reinsertsen, is a system condition that combines high capacity utilization with low throughput. It can be compared with traffic on a highway. Throughput (or flow) can be measured in number of vehicles per hour, which is the multiplication of density (d: number of vehicles per kilometre) and traffic speed (v: average kilometres per hour). Density decreases with speed (for instance linear like d=a-b*v for some constants a and b). The product (flow) becomes f= av-bv^2, which is a parabolic function as depicted in the graph. It has an optimum. On the right of the optimum, there is not enough traffic, but it is inherently stable: when density increases and speed drops, the flow increases, reducing density. On the left, there is too much



traffic and this side is inherently instable: when density increases and speed drops, flow decreases, which further increases density…. ending in a state of heavy congestion.

It is like in computer systems, where too many processes all of a sudden make the computer is only swapping processes and no single process makes any progress anymore. And also from telecommunications we know it is better to throttle Work In Progress (WIP) to avoid congestion.

It is clear that the more uncertainty there is in the system, the better it is you stay on the stable side of the curve.

Congestion easily happens in software development

There are many instable situations in product development which may lead to congestion.

There are product marketing people, who believe product development will produce more output when the demand is higher. In reality, higher demand increases work in progress, lowers productivity and creates even more demand from product marketing.

Or adding resources, while the new resources take more time from the existing resources to get trained, lowering productivity and creating even higher demand for new resources.

Or deliver the feature with insufficient quality, leading to more work to repair the defects and lowering the productivity to deliver new features.

Or accumulating technical debt to deliver a feature sooner, at reduced quality, without a plan how to pay back the debt, resulting in more difficulties to implement new features, lowering productivity and creating more pressure to take on even more technical debt. There may be good reasons to temporarily accumulate technical debt. Reducing, however, the quality of the execution without a reason to speed up, is called technical crap, and is hard to justify.

Moreover, the intrinsic uncertainty, we often face in of software development, makes you better stay on the stable side of the curve, as passing the tipping point will rapidly kill productivity.

How can you detect congestion in software development?

I have been able to observe organisations moving from highly congested, to very productive and back to congested again.

Measure throughput

You can measure throughput and you can notice a drop in throughput. Variations in throughput happen all the time, but when you are on the stable side, you can handle these variations without ending up in an instable situation. Most development organisations do not measure throughput. Luckily, there are other signs that should warn senior management for a potential risk of congestion.

More resources are often counter effective

The first red flag is a request for more resources. More resources will almost always drop velocity. It often takes half a year to build a new team and achieve similar performance levels than experiences teams. Teams that worked together, but just have to learn a new domain, may be faster, but there will still be a velocity drop. Plans that do not take this drop into account should immediately trigger an investigation to see if congestion in the team is not the reason for a request for more resources.

Decreasing quality often signals congestion

Of course, another indicator is the quality of the deliveries. An increase in number of open defects or an increase in number of leaked defects can be a good warning the organisation gets congested.

Another good indication for congestion is technical debt. There are tools on the market to check the quality of the code and the compliance against software quality standards like ISO 25010. A decrease of code quality is a strong signal for congestion. But also, the number of branches may be a signal: when the number of branches starts growing, you better be check for congestion.

Transparent reporting

There was this team that heavily suffered from congestion. We totally re-organised the team, focused a lot on quality and created full transparency on progress, risks and issues, both internally and to all customers. The top management was very enthusiastic about the substantial improvement in productivity. We were even asked to help deploy the practice to other parts of the company. After some time, another manager took over. This manager was an excellent product manager with good market insights but less experiences in managing big software teams. He changed the reporting systems and became less transparent to his customers. Later we found the team heavily congested again. We should have been warned when the reporting system was changed, as later it turned out his team had produced CFDs clearly indicating big problems, where the new manager continued to report things were under control.

In this case, it would have helped if I as the managers manager had spent more time with the teams directly. They are often more aware of the congestion than the managers. You can also read this from team happiness indicators. In the end, I should have been warned by the change in reporting but the real mistake I made was to underestimate the risk of insufficient software leadership skills: a manager who does not appreciate the value of CFD will not be able to detect and avoid congestion.

Ask your customers

Finally, ask your customer. In the case described above I got really concerned when customers started reporting on lower quality and lower output. They started using the same words as used before we reorganised the department: “can’t-do mentality”. Although I agree it is important for a product owner to be able to say NO, I have noticed that in efficient organisations there is always room for unforeseen things. That is the benefit of Agile. That means, customers will understand the product owner’s NO, when there is often also room for a small YES. When the answer is always NO or DIFFICULT, without even looking at the bang per buck, you should be warned the teams are probably overstressed and the organisation is congested. So, it is worthwhile to regularly check with your customers.

How can you avoid congestion in software development?

Plan a buffer at the end of the critical path

In 2014, I met Dean Leffingwell, the founder of SAFe, the scaled agile framework for enterprises[1]. One moment I asked why we need to have the last sprint in the iteration to be of a different nature. He responded by asking me two questions, which said it all:

1. Did you read Goldratt on The Critical Chain? This is the buffer at the end of the critical path
2. Did you read Reinertsen on Flow? This is the free capacity to deal with the uncertainty.

He just mentioned two of my favourite books and it explained all. How more exciting can it be? Later he added many references to Reinertsen on the SAFe website, which substantially strengthened the theoretical basis for the model.

You can only deal with uncertainty if you do not plan all capacity for critical (committed) features. You should reserve a buffer, which you can use for might have features, for planning, for innovation, whatever. If you hear from the teams they don’t have time for innovation or they don’t have time for a buffer, you have a clear signal of congestion. Once in a while, teams may need the buffer for development of committed features, which is not a problem. It is a problem when it happens every time.

How big should that buffer be. I asked several university professors on systems dynamics. They all come with a rule of thumb that you should not plan capacity over 80%. I believe it should be related to the level of uncertainty, but I have not yet found the mathematical model.

[1] www.scaledagileframework.com

Drive cycle time, WIP and quality

Like in any organisation, focus on cycle time helps. I once talked to a judge, responsible for a court department, and he mentioned the public complained about long lead times. It took 14 month to change an alimony. When I asked how much real work was involved, he mentioned on average 30 man-days. Methods like quick response time manufacturing focus on lead time to make organisations more efficient. It also helps here. Complexity is dramatically reduced by smaller cycle times and you achieve smaller cycle times by focusing on WIP. And not only the feature backlog but also the defect backlog. We introduced a rule that when defect WIP was above its limits, we would drop feature development. The impact of a quality focus on productivity is often undervalued; it increases the uncertainty in the execution.

Create transparency and listen to the teams

As said, teams often feel the congestions first. Systematic overtime, delivering features that did not yet meet the definition of DONE or not even had a definition of DONE, etcetera. In the team mentioned above, at some moment in time, they spent two months after every release to mature the release. And that with 2-month release iterations. Does not really sound Agile, does it? With proper transparency in place, everybody would have noticed and corrective actions could have been defined.

Have strong management in place

In the end, it always boils down to stronger management. Management that can explain the need of unplanned capacity to their management, management that dares provide the right transparency, management that understands the concepts of WIP and congestion, and management that is capable of taking the right corrective actions.

How can you repair congestion in software development?

Pull the brakes on feature development

In traffic, congestion is often repaired by reducing the number of cars on the road. In severe situations of congestion, that seems the only way out. You can’t increase the number of lanes. Even worse, creating more lanes, will further complicate the traffic situation and lead to more congestion.

The same holds for software development. Once congested, adding resource does not help, not does just pushing the delivery dates. Reducing the scope is the only thing that can lower the WIP and fight the congestion. That is not always easy, as commitments have been made. But once congested, the commitments will not be delivered in time, so pretending you can deliver on your commitments is even worse and not a sign of professional behaviour.

Break with the past

I think we have been lucky with getting rid of the congestion in the software team mentioned before. There was hardly any output and the output had substandard quality. The customers suffered in their projects and faced severe issues in delivering. Top management decided to cut almost all the dependencies of the team in the organisation. The team was allowed to start again with only one branch, a new way of working, a strong quality focus, change management and only one customer to focus on. These were the right conditions for success and one year later the team delivered on what the CEO mentioned to be the best quality, most reliable and most transparent execution ever. Only after that more customers were added; before, these other customers had to deal with the problems themselves in their own branches. This sounds very inefficient, but they anyway already dealt with all issues themselves as the output of the team was almost zero. So, not a loss of efficiency, but a brave decision to improve the future.

Replace management

Repairing congestion requires even stronger management than avoiding it. Don’t even consider asking the same manager that let it happen to also repair the misery. Very often you also want to give a strong signal to the organisation things are going to change. You often have to re-establish the confidence in management, as the people on the floor knew it and suffered for a long time.