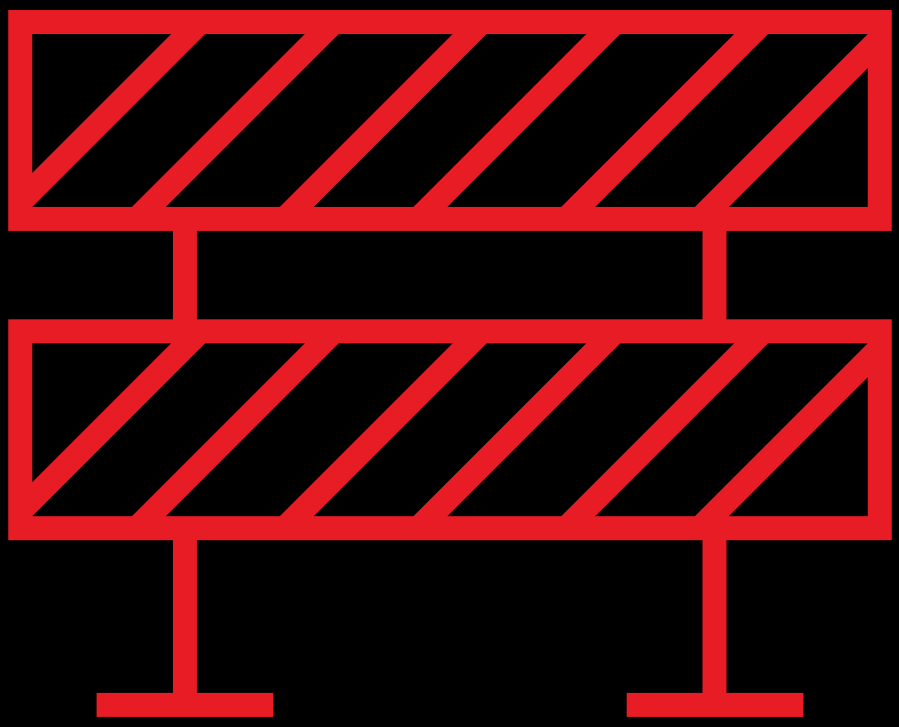
VFQ Value Flow Quality



Work in progress

by emergn

This publication forms part of Value, Flow, Quality® Education. Details of this and other Emergn courses can be obtained from Emergn, 20 Harcourt Street, Dublin, D02 H364, Ireland or Emergn, 190 High St, Floor 4, Boston, MA 02110, USA.

Alternatively, you may visit the Emergn website at http://www.emergn.com/education where you can learn more about the range of courses on offer.

To purchase Emergn's Value, Flow, Quality® courseware visit http://www.valueflowquality.com, or contact us for a brochure - tel. +44 (0)808 189 2043; email valueflowquality@emergn.com

Emergn Ltd.
20 Harcourt Street
Dublin, D02 H364
Ireland

Emergn Inc.
190 High St, Floor 4
Boston, MA 02110
USA

First published 2012, revised 2021 - printed 16 July 2021 (version 2.0)

Printed and bound in the United Kingdom by Apple Capital Print.

CONTENTS

INTRODUCTION

We'll begin by recapping the idea discussed at the beginning of the Attacking Your Queues session. Queues exist in every industry – but the more visible they are, the more likely we are to do something about them. In any type of manufacturing work, the issue is very clear to us. A pile of circuit boards sitting around gathering dust will drive a manager into furious activity, because he knows that circuit boards quickly become redundant. All the value invested in these components will go to waste unless they can be turned into a finished product and shipped to a customer. The staff care too – they can't work on anything else until they get rid of all this clutter.

Unfortunately, in software development it is easy to ignore a big pile of partially done work sitting in a queue. It could be a requirement specification, an amount of code (working or not), or even a complete, finished product which has not yet gone live. This partially done work represents an investment which is not yet creating any value.

Traditional manufacturing would have called this 'inventory', a term which includes the piles of components, half-finished products and products that are finished but not yet shipped. In software development we normally refer to it as 'work in progress' or, because we are in an industry suffering from advanced acronym addiction (AAA), it is also known as WIP. The term covers not only the work being actively processed, but also work which is sitting around waiting. Obviously we would like to minimise the amount of half-done work and get our ideas realised and launched so that we can make some money from them!

This session considers how we can help move work more quickly through the cycle in order to release the value that it represents to the company. The idea of managing WIP is intimately connected to the two sessions on Attacking Your Queues and Batch Size Matters – so much so that changes in one tend to have a major impact on the others.

By the end of this session students will understand:

1. Work In Progress (WIP) in software development.
2. The principles and benefits of limiting WIP.
3. How to monitor and manage WIP at a personal, team and project level.
4. The potential downside to temporary or permanent WIP limits.
5. Alternative methods of WIP control: throttling, purging, shedding scope and phasing.

1 PULL, DEMAND AND WIP

Larman and Vodde state in Scaling Lean & Agile Development:

> "Product development people do not really see and feel the pain of their queues. Yet, they are there. Queues of WIP – information, documents, and bits on disk."

Activity 1: What's your Work In Progress

This activity should take no more than 10 minutes. It forms the basis for further activities within this session so ensure you keep all your notes.

Select a team with whom you are working (at least 3 people). A project team you are working in now and for a few months to come would be a good choice. Alternatively, you could also do this with your functional team.

Working together, use Post-its to note down the WIP that exists at the moment. Begin with what you are working on in this iteration or phase of the project – normally a period between 1 week and 1 month. Try to include everything, not only features and fixes, but also other tasks, from reinstalling software, to submitting purchase orders. Use one Post-it per task you identify.

Tip: Don't go too granular (collect expense sheet, fill it in, hand it to boss for sign-off), but don't go too macro either ("housekeeping" probably needs to be broken into component parts – status report update, team expenses, order new hardware, etc.)

Now create a simple visual board. Ideally pick a whiteboard that you can keep for a few weeks or a flipchart page on a wall. Draw three columns and label them as "To Do", "In Progress" and "Done".

Put all the notes you have created in the middle "In Progress" column.

To Do	In Progress	Done
		

Over the next couple of weeks keep using the board and observe your Work in Progress. As you complete each tasks move the Post-it over to the "Done" column, when new tasks arrive add Post-its to the "To Do" column and then move them over to the "In Progress" column as you start dealing with them.

Commentary:

What did you discover? In a large number of cases when teams are asked to list all of their WIP they are shocked by just how much they are working on and furthermore how much else is in progress, but not actively being worked on.

We have established that controlling queues can lead to faster delivery. One of the best ways to manage them is to constrain the amount of work in progress at any one time. At the moment, Don Reinertsen suggests in his book The Principles of Product Development Flow, only 3% of developers use this technique. Those who do know the idea have often heard of it referred to as the Kanban method, rather than as a 'work in progress' constraint.

Although kanban was originally developed for use in traditional manufacturing, many of you will have come across the term in product and software development. Indeed, the idea has some major converts, who believe that kanban can cure many of the woes caused by queues.

The principle of kanban is to link feedback between the different areas of WIP (or queues), so that only work which can be processed enters the system. This is why it is called a "PULL" system – a process downstream pulls in a new piece of work, as opposed to the traditional system where an upstream process pushes a piece of work on to the next process.

Corey Ladas summarises the efficiency of pull in his book Scrumban as: 'Don't write more specs than you can code. Don't write more code than you can test. Don't test more code than you can deploy.'

Let's imagine we're making cakes on a fairly large scale – as a wedding cake specialist, perhaps. Our oven has six shelves and can hold six cakes at a time. Wedding cakes must be baked slowly at low temperatures, often for 4 hours or more. Despite having the best oven on the market, temperatures do vary throughout the oven, meaning some of our cakes will be ready sooner than others. That means there will be variation in our process – sometimes we will have space in the oven for two new cakes, sometimes four... it just depends.

When a wedding cake has been baked it can last for months, but the cake mixture (containing raw eggs) goes off fairly quickly. We don't want new batches of cake mixture to arrive when we don't have cake-tins to hold it. Nor do we want tins of unbaked mix sitting around waiting for three hours until there is space in the oven.

Figure 1. Cakes cooling and tins becoming available

Of course, we could just walk into the mixing kitchen and say to the cook, 'make me another four batches of mixture', but instead, because we know all about kanban, we invent a system that stops the upstream process (mixing) from over-production, by signalling when there is capacity.

As a cake comes out of the oven and is placed on a cooling rack, a tin becomes available. The tin is sent back to the mixing kitchen. The sight of an empty tin lets the cook know when to prepare another batch. When no tin arrives, he doesn't make cake mixture. If three tins turn up, he makes enough for three.

This is described as a pull system, but in reality, it is about limiting the work in progress to the system's immediate capacity: in our example, limiting the cake mixture to the number of empty tins and thus available shelves in the oven.

David Anderson writes in his book Kanban that, 'An interesting side effect of pull systems is that they limit work in progress to some agreed upon quantity, thus preventing workers from becoming overloaded.'

Activity 2: The benefits of pull

This activity requires between four and six players and will take approximately 15 minutes.

Objective: You are in the business of building paper houses and selling them to a new and emerging market. Make your company as profitable as possible in two rounds.

Materials:

- 2 – 3 pairs of scissors.
- 2 sticks of glue.
- Coloured index cards (you'll need 3 different colours).
- Envelope.
- A pen.
- A timer.

Preparation:

In order to build a house you will need: a plot of land (a single index card – we chose pink), the house structure (a shape of a house cut from an index card – we chose yellow), door and windows (here we got creative and chose blue and green!).

You will be selling four different types of houses:

- Type 1 – square door, 2 square windows
- Type 2 – arched door, 2 round windows
- Type 3 – square door, 2 round windows
- Type 4 – arched door, 2 square windows

Before starting the rounds build one or two houses to agree what your houses will look like.

You also need to simulate a market for the houses. To do this, create 10 tickets for each of the 4 types of houses on separate pieces of paper, so you should have 40 altogether. Put the tickets in the envelope and shuffle them around. You'll draw from this envelope later.

Rules:

1. Every house sold earns you $150,000.
2. Any materials you have remaining at the end of a round will be deducted from your overall profit using the numbers below:
   - $100,000 for a house that has been built but not sold
   - $75,000 for an unfinished house (missing windows or doors)
   - $50,000 for a plot of land on which you've laid foundations (the house structure card has been glued to the plot of land card)
   - $15,000 for every unused door or window.

Play:

You will play two rounds, each will last 5 minutes.

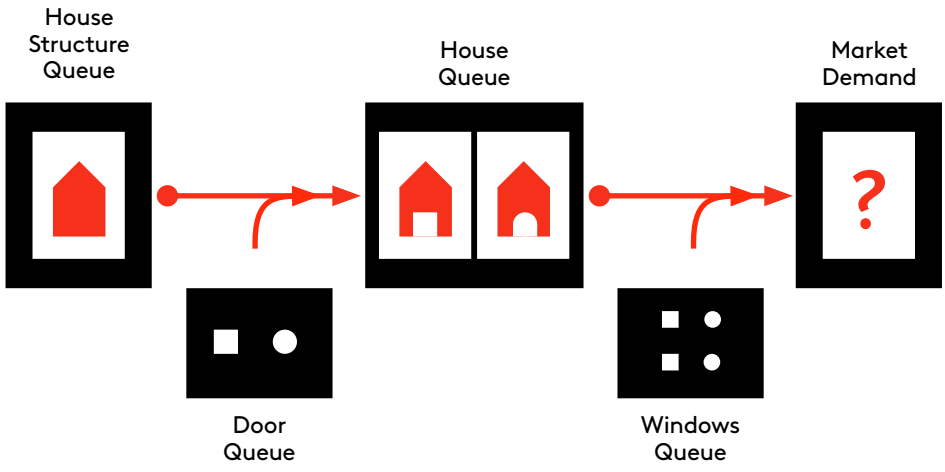
Round 1

1. In this round, each team builds as many houses as they can, aiming to make a good profit on the sale. Decide on how best to achieve that, bearing in mind there are four different types of houses you could sell.
2. Select an index card for a plot.
3. Cut the house outline from an index card and glue it to the plot.
4. Decide on the type of house to build, then draw the door and windows the house will need.
5. Cut and attach the correct door.
6. Cut and attach the correct window.
7. When the round is over cease building immediately. Leave any unfinished houses as they are – do not complete them.
8. Count the number of houses you completed. Draw a corresponding number of tickets from the envelope. If you have a house type that matches the demand ticket pulled, you sell it.
9. Now calculate your losses and gains according to the rules above.

Round 2

Before we begin this round, put all the tickets used in the previous round back in the envelope.

Now reorganise your process. Your assembly line should look as follows:



- The House Structure Queue – contains one house structure placed on a plot.
- The Door Queue – contains 1 square door and 1 arch door.
- The House Queue – contains 2 houses – one with a square door and one with an arched door (and no windows).
- The Window Queue – contains 2 round windows and 2 square windows.

Once the queues are ready (and are manned by at least one person) begin the round.

1. Pick one ticket from the market envelope.
2. According to the required model pick the correct windows from the Windows Queue.
3. Attach them to the house from the House Queue and release it to the market.
4. Replenish the Windows Queue and the House Queue.
5. To refill the House Queue you will need to pick a Door from the Door Queue and a House Structure on a Plot from the House Structure Queue.
6. Make sure all queues are full.
7. Now you're ready to build the next house. Repeat until your time runs out.
8. When the time is up, tally up your earnings again, including the cost off all the material that remains in the 4 queues.

Compare your results from Round 1 and Round 2.

Commentary:

In Round 1 we demonstrated a Push system, whereas in Round 2 you used a Pull system. Round 1 saw you assigning requirements and building houses without any knowledge of the actual demand. It's likely that there was a significant amount of both unfinished work and unsold houses at the end of the round, because of the mismatch between the requirements you selected and the actual demand. In comparison, in Round 2, your actions were directly driven by the demand, which resulted in fewer unsold and unfinished houses, or less waste.

By pulling materials from the queues and only replacing this inventory on-demand, we limited the total amount of work in progress.

2 MAKING WIP VISIBLE

The important part of the rather simplistic cake-making kanban or house-building kanban described previously is that we have made the process visible. If you buy your coffee from a Starbucks store, you witness the kanban system at work. The operator at the cash desk writes your order on a cup: skinny latte. The cup is passed to the barista who then fulfils the order. There is no need for her to ask which size – she has the correct sized cup in front of her. There is no need to match a ticket to an order – it is all on the cup itself.

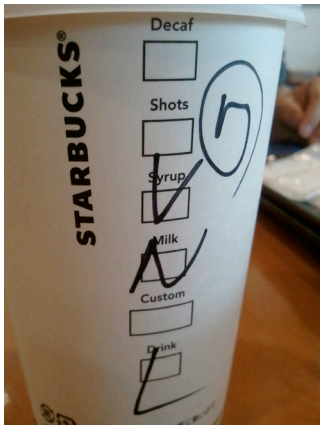


Figure 2. Starbucks kanban system in action

There are two major benefits: firstly the ordering/paying and coffee-making systems have been decoupled reducing the need for potentially time-wasting or confusing communication issues; secondly, we have made WIP visible. If thirty cups are piled up in front of the barista it becomes obvious that we have a problem – spare capacity needs to be switched to the coffee-making system in order to attack the queue.

In order to manage our queues and WIP we need to make the information that controls it visible.

Figure 3. A Toyota kanban colour-coded card

Lean manufacturing companies who developed kanban do this very well. Toyota works hard to ensure that its plans are big and simple. Designed to be understandable at a glance and from a distance, they are displayed on walls with colour-coded cards that can be touched and moved about. Indeed, kanban literally means 'signboard'.

By contrast, software development typically measures its progress on a spreadsheet or project plan on the Project Manager's machine. Filled with hundreds of tiny boxes, often not shared with the team as a whole, and sometimes tracking a planned schedule rather than what is actually occurring, such an approach is the opposite of the kanban system. Indeed, a system like this typically ends up generating more work – the project manager has to check with team members in order to update his plan or spreadsheet as well as feeding back at team meetings, and meetings with senior management or customers, on how things are progressing.

CASE STUDY: War-game simulation

There can rarely have been such an appropriate example of the help that visible planning can give than the 'war-game simulation' that was being played by the staff of Fifth Panzerarmee during the winter of 1944.

The Germans were attacking allied forces during a period of bad weather, hoping to reach the port of Antwerp and force the Allies to sue for peace. According to German General Friedrich J. Fangor, the staff had met in November to game defensive strategies against a simulated American attack. They used a map table and moved around physical figures to represent army units – in particular they focused on the Hürtgen area. While they were doing this, reports began arriving of a strong American attack at exactly that point. General Field Marshall Walther Model ordered the participants to continue playing. Instead of rolling a dice to simulate success or failure, however, they used messages they were receiving from the front as game moves.

Figure 4. An operation room during World War II

The visualisation meant an unusually clear strategic response. When the officers at the game table decided that the situation warranted commitment of reserves, the commander of the 116th Panzer Division was able to turn from the table and issue as operational orders those moves they had been gaming. The division was mobilised in the shortest possible time, and the American attack was fought off.

Source: War Games: Inside the Secret World of the Men Who Play at World War II, Thomas B. Allen

2.1. How do we make software development WIP visible?

Kent Beck in his book Extreme Programming, describes the ideal for information dissemination:

> "An interested observer should be able to walk into the team space, and get a general idea of how the project is going in fifteen seconds. He should be able to get more information about real or potential problems by looking more closely."

The purpose of this is to share information effectively throughout the team – it is sometimes called 'visual management', while the physical boards and displays have been somewhat pretentiously named 'visual radiators'. Rather than one person being responsible for keeping progress updated on a spreadsheet and then disseminating this information via emails or at meetings, the team has shared ownership of information. The result is a greater sharing of responsibility for managing the team or department's work and more transparency for management, other teams and within the team itself.

The information display required depends on the team and the project, but the most common is a task board. This simply shows the list of activities being done at the moment, work on hold and work coming up: it can range from bugs and legacy issues to user stories and upcoming live tests.

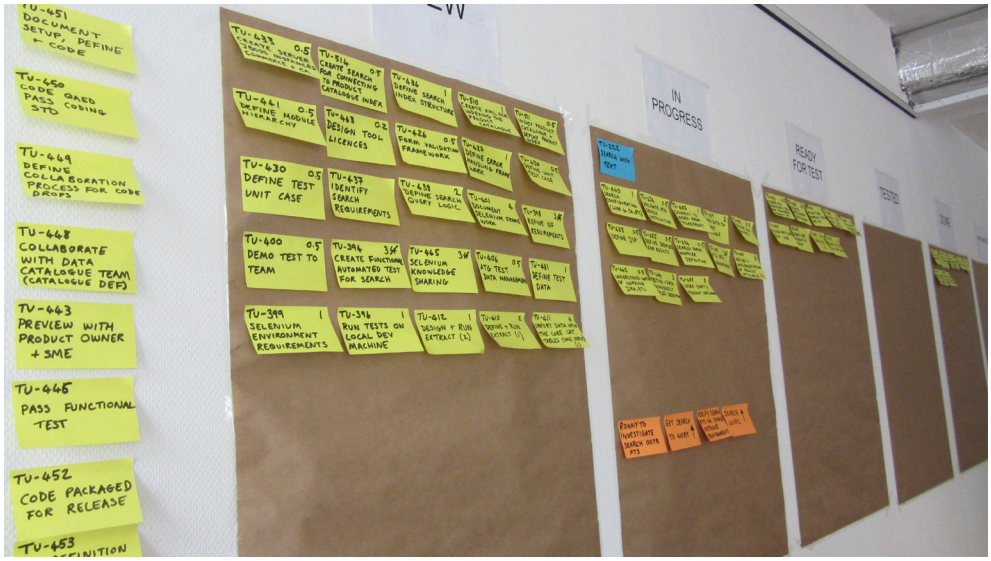


Figure 5. An example of a kanban board in action

Benefits and pitfalls of visual task boards

A good visualisation of the team's work means that you can cut down on progress reports and updates. It makes it easier to spot emerging queues whether for individuals, processes or departments, allowing the team to revise estimates or consider appropriate action (don't worry, we're coming to the action...).

Of course, any system only works if people use it and find it useful. If updating the board is seen as a painful chore then it becomes worse than useless. This can be the case if an organisation signs on to the idea of visualising WIP but then continues to rely on the miniscule project plan or spreadsheet in parallel to the task board. Another duplication of effort occurs if information is requested in a manner that does not fulfil its primary purpose of helping the teams. For example, if an email has to be sent that summarises the board to senior management, rather than a manager dropping in to look at the board.

The board also preferably needs to be physical: Post-it notes can be moved around, headings can be rubbed out and re-written... if the board becomes a software version with a display print-out, then you can often be back to the problem of not fully sharing ownership and responsibility for the information. We understand that for some teams, it will not be feasible for the board to cover absolutely everything or to be used across geographically distributed offices. There are numerous tools available that provide shared online task boards, user stories and project backlogs visible to all users, where you can still move about, reprioritise and add notes to each item. If properly committed to by the team, they can work, although there is still a tendency to suffer from their lack of physicality. A fancy graph drawn on the screen provides an illusion of control that can be dangerous – especially if it turns out that only half the team are really monitoring the tool.

A task board represents the flow of work, which means it needs to be capable of change and flow itself. We would stress that the more simple and dynamic a board is, the more useful you are likely to find it. We often see companies with teams who have become so enamoured with their whiteboard tape and colour-coded Post-its that the system has ceased to be flexible. 'No, I can't add another column to represent testing,' snaps the team lead, 'that would mean re-measuring and redrawing all the other columns on the board.' There's no need to worry about perfection – you can start with a board as simple as: To Do; In Progress; Done. As you get more familiar with it and gain further understanding of your work, you can improve the board to represent your flow of work more precisely.

Finally, the point of having a board that can be changed on a daily basis to show the flow of work is that it must be changed on a daily basis to reflect the flow of work. This requires commitment and discipline from the whole team – which is why when successful, a task board speaks of a properly-welded team, while nothing speaks of issues like a task board which is a month out of date...

Figure 6. A signboard outside a factory

Have you seen signboards outside major factories announcing: "57 days without an accident"? The board has to be updated every day – by hand. An electronic display would not suggest the same thing – that every member of the factory is focused on, and responsible for, safety. Similarly – you can suspect a company is spouting ideals it does not believe in when the sign board is up without anyone changing the numbers from day to day – or worse, without any numbers at all.

3 SELF-MANAGEMENT AND WIP

Remember that the point of the visual sign of the empty cake tin was to tell the mixing cook to make some cake mix, and the point of the Starbucks coffee cup is to tell the barista what to make. Compare this to the way a top kitchen works – orders come down as 'tickets' and the head chef calls out the orders to each station (2 chicken, 3 beef, 1 vegetable pasta). The head chef is the key communicator and each individual chef needs to remember what orders he has in hand (his WIP) and their timing. Not only does the system often go wrong, (where would we be for television entertainment nowadays without chefs shouting at one another?), but it takes up a very valuable resource (the head chef) as the coordinator and manager. If the head chef were able to institute a visual system – setting out the correct number of plates on each work station, he might suffer less stress and free up more of his own time. Note that this would remain a push system – woe betide the chef who told Gordon Ramsay, 'I only have capacity to produce another two meat carpaccio'...

A full visualisation of WIP enables easier and better self-management and facilitates pull rather than push. Rather than having work assigned to them by a manager, team members are able to commit to take on a number of tasks from the task board. The real benefit of this is to stop individuals from becoming overloaded, and thus helps to limit unseen queues and the resulting delay.

To go back to our head chef example for a moment, if the meat carpaccio is especially popular, all the other chefs will need to work at the pace of the meat chef. Since the whole order must be served together (dishes that have been waiting at the pass for too long are sent back to be redone, creating longer delays and waste) – the other chefs should begin their dishes only when they are sure that the meat chef is ready to begin on that order. The difficulty of making the restaurant ticket system work in this kind of visual, pull way, helps us see why software development departments without visual systems also end up discovering bottlenecks only after a queue has already built up.

In a self-managed visual system, team members (or teams) only 'pull' the quantity of work they can deliver. This is, in essence, a limit to the work in progress in the system. Agile methods depend on this pull approach – taking on features from the backlog that the team commits to in that Sprint only. Finally, because a task should be completed before taking on a new one, it also stops a fragmented approach of working on several things at once (with the attendant switching cost).

Activity 3: Policies help limit the amount of Work In Progress

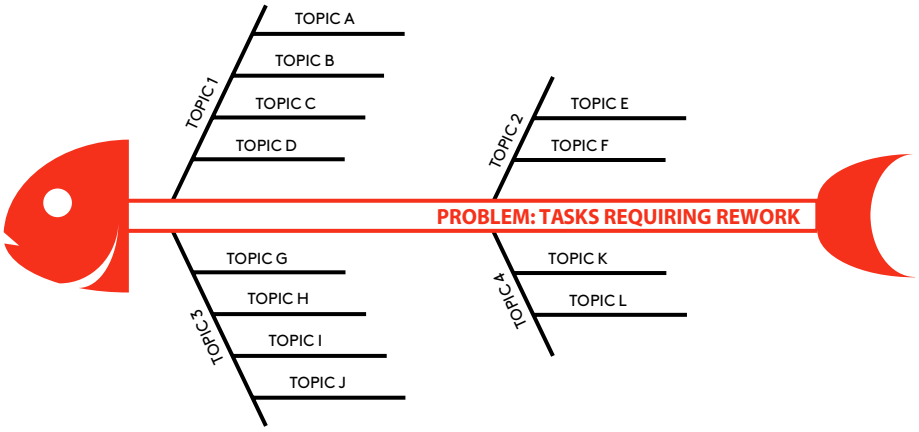
This activity should take about 20 minutes.

Once the board you have prepared in Activity 1 has been established for a couple of weeks, gather the same group of people together.

In Activity 1 you identified all the tasks that constituted your Work In Progress. We will now examine these items more closely.

There are many reasons why work gets held up within a team or process. One of those is that tasks previously thought to be 'done' come back because something was missed first time round. This is called rework. Not all rework is harmful – it might be that we have received feedback from a customer telling us how our product could be improved (that would be good!). But problems that ought reasonably to have been caught the first time round are bad.

Consider the activities that you have observed moving across your board. You are going to explore the potential causes that may lead to a need for rework, using a tool known as a 'fishbone diagram'.



On a large sheet of paper draw a long arrow across the middle and label it "Problem: Tasks require rework". This is the 'backbone' of the 'fish'. Draw spurs coming off the 'backbone', one for every likely cause of the problem and add a label at the end. Keep exploring the main causes, adding additional ideas and digging deeper - there may be several possible causes behind each problem. Indicate each new cause as a labelled line connected to the main spurs. Don't worry about writing the same things multiple times. Capture all your ideas in the diagram as people think of them.

Once you have gathered some insights from the fishbone diagram identify three or four main causes of rework affecting your current process.

Use the last 5 minutes of your time to write a simple policy (in the form of a checklist): what criteria does a new piece of work need to meet in order for you to start working on it? Write these criteria with a view to minimising rework in the process.

Use this checklist over the next couple of weeks in conjunction with your board. Apply it to any tasks that you are about to move from "To Do" to "In Progress".

Note what effect this has on the level of your Work In Progress. Has the policy itself caused any problems?

3.1. Monitoring WIP

We have stressed the importance of measuring and managing queues. It is also important to monitor the tasks within the queue. Specifically we're looking for 'Outlier' tasks. They are tasks or items in the process which fall outside average delivery timescales or are not completed within an iteration. These have the potential to place unpredictable demands on the flow which can then delay other tasks. To do this we would usually expect to track progress against an average cycle time. When an item has considerably exceeded the average time, we would then escalate it as a problem. For example, if most user stories (the common method of tracking work in Agile) are being completed in 5 days, but one user story remains on the board for two weeks, we might start to become anxious and decide to investigate. Does it contain an unexpected technical challenge? Is the story too large and does it need to be better defined?

Even without an objective measure, most teams know intuitively when an item has been hanging around for too long. Rather than letting the item continue to sit on the board, or passing it from person to person, a mini-team can assemble to crack the problem – this is sometimes referred to as swarming.

Figure 7. Swarm of bees overcoming a large problem!

Another reason that work can hang about stems from Parkinson's Law: works expands to fill the time available. Some tasks could be never-ending – try getting a writer to admit a book is ever actually finished, or a designer to stop tinkering with colours and images. Similarly a developer can get so intellectually buried in a task ('I will solve this defect being caused by a race condition even if it kills me') that she will work on a solution long after the sensible decision is to drop the problem. Monitoring the length of time taken by tasks and keeping their associated costs in mind enables us to call a halt to work that threatens to negate its own value. A continual assessment is a far better way of doing this than the usual method within software development where we assess a feature or project's value and cost after it has been delivered.

4 LIMITING WIP

Think of when a large jam forms on a motorway. Signs upstream announce major delays, encouraging motorists to find alternative routes; traffic bulletins on GPS navigators or the radio suggest turning off the motorway. In extreme conditions, traffic controls actually block access to the motorway in order to stop more cars joining the queue.

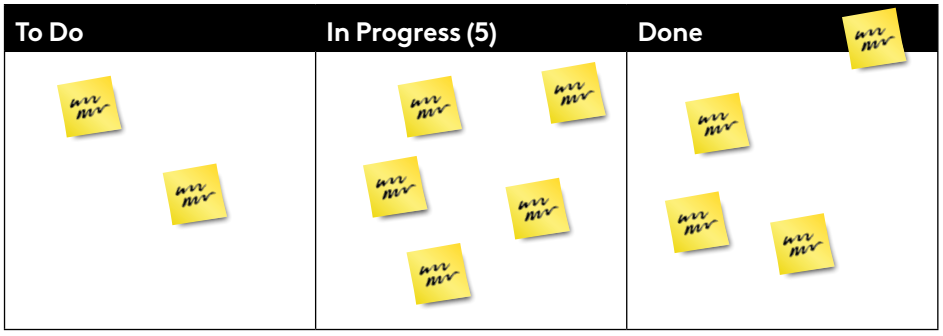
In software development terms this can be thought of as blocking or rejecting demand – sometimes this may be a minor adjustment, at other times it can work at the macro level of not taking on entire projects.

Activity 4: Implementing WIP limits

This activity will require about 20 minutes of preparation and then at least 6 weeks to monitor progress.

Continue using the board you created in Activity 1 and working with the same team.

Now it's time to apply WIP limits to the work you are doing. To do this: consider the number of people working on the tasks in the "In Progress" column, multiply the value by 1.8 and round the result down. The result is now your WIP limit. Write it in the "In Progress" column heading.



You can't start more tasks than the indicated WIP limit even if it means people have to wait for others to finish, or share a task in order to help a colleague finish it more swiftly.

This activity prepares you for Activity 6, where you will be asked to run a session with your colleagues at regular intervals throughout the 6 week period that you are observing your workflow. Keep a diary of your progress. Make sure that you record: the state of the "To Do" queue and how it affects your interactions with any upstream processes. How busy are people working on the activities? How are tasks released into the downstream processes?

Commentary:

The multiplier of 1.8 has been developed through experimentation with several companies. In general we have discovered that 1 task per person is too low, causing problems for the team's efficiency, while 2 tasks per person is too high, failing to encourage interaction and continuing to create bottlenecks. You will have a chance to adjust the figure later for your unique situation, but we urge you to stick with the WIP limit for at least 3 weeks to really understand its impact.

4.1. The benefits: cycle time

There is an obvious relationship between the amount of work in progress in the system and the likelihood of queues forming: the more WIP, the more likely the queue. The reverse is also true: the fewer tasks, the less chance of a queue. And what do we know about a lack of queues? That's right – low queues mean fast cycle times, zero queues means the fastest possible cycle time.

One way of reducing the queue is to stop adding more tasks to it. The result of limiting the amount of activities being worked on at any one time means that the work to which we have committed will get done faster. Rather than spreading ourselves between three or four projects, we commit to only one and the project as a whole is finished more quickly. If you are serving just one customer, you will process their request faster than if you are serving two at once. This is true even if each customer is not taking up 100% of your time.

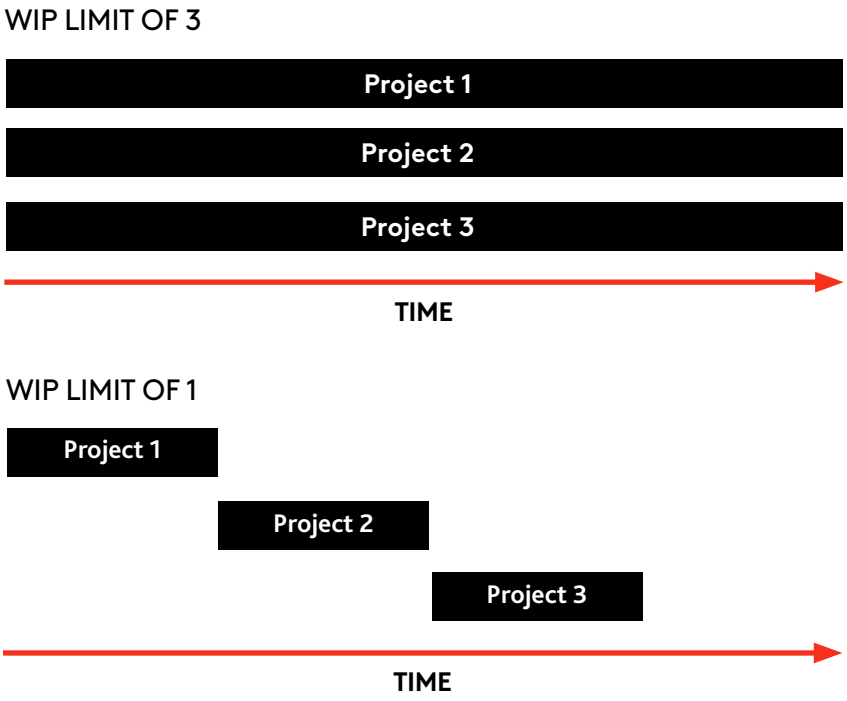


Figure 8. Reducing WIP to get things done faster

If a department is working on four projects which are all due for completion in one year, the company could reduce cycle time by working on only two of them – naturally picking the two of higher value to the company. These would then finish in six months, could be launched earlier and begin generating revenue earlier. Now the company can decide to start working on the remaining two. These benefit slightly from a later start – our information and technology may have improved. More importantly, these two projects may have been superseded by ideas in the pipeline which are more valuable.

CASE STUDY: XIT Team at Microsoft

The XIT Team, a vendor team based in India, were responsible for developing minor upgrades and fixing bugs for about 80 cross-functional IT applications used in Microsoft. Although it produced good quality results, it had a reputation for terrible customer service, with a 5-month lead time on change requests and a growing backlog. All this despite the fact that the average request took only 11 days of engineering time. There were good reasons for why the team was being overwhelmed – it was not that they were slow or poor developers. Problems included the random manner in which requests arrived, the need to provide accurate estimates which consumed about a third of capacity and small changes which were given priority, thus derailing the predictability of other work.

Several big changes were instituted – the most important of which was to limit WIP. The first element of limitation was to permanently reject some demand – the estimating work – in order to recover this capacity for development work. Various measures were introduced to help with the budgeting implications of a lack of exact unit costs which essentially allocated total capacity between the different business units using the XIT team. Secondly, the team had a WIP limit of 1 item per developer, and 1 per tester, with a queue between the development and test functions to even out variance.

The results were very impressive. Requests were processed within a new 25-day guaranteed SLA. The product managers were selecting the most important requests to fill however many spaces were available (the WIP limit ensuring that the team pulled this, rather than having it pushed). By not having to continuously re-order and prioritise the backlog this actually saved them time as well. Finally if a request was not deemed sufficiently important to have made it out of the backlog within 6 months, it was purged. If it was still deemed important it could be resubmitted. After seeing the success of these changes, the team was able to tweak its capacity (one less tester, one more developer) in order to remove a bottleneck in development. With additional capacity added (2 more people), throughput increased beyond demand meaning that the backlog was eliminated entirely. The team had reduced lead time to 14 days for a request still taking an average of 11 days of engineering time – that is a department running with very low levels of queuing. The graph overleaf shows the quarterly throughput overlaid with unit cost.

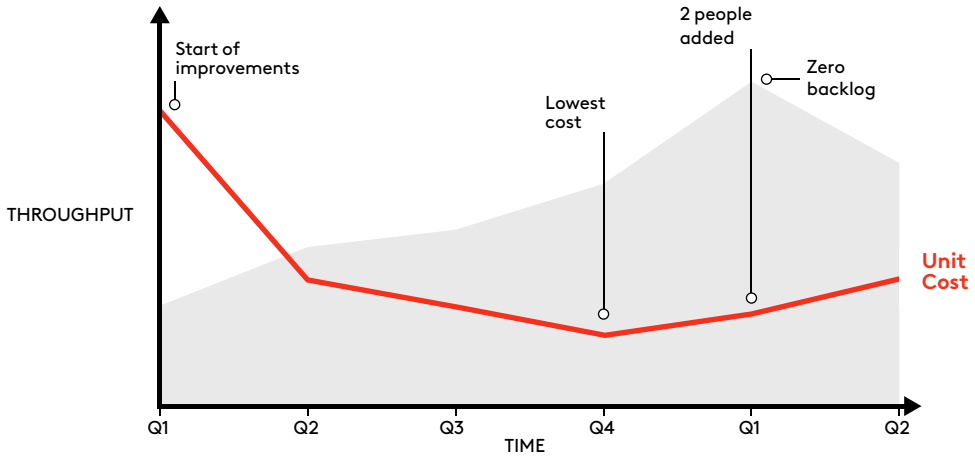


Figure 9. Quarterly throughput overlaid with unit cost

Note the graph shows that the fastest development cycles include a slight increase in unit cost from the maximum throughput point, but that this unit cost is still much lower, and the efficiency much higher, than running with a long queue.

Activity 5: The name game

This activity should take you only a few minutes. Ask a colleague to help you track your time.

Pick six people you know, either your friends, family or colleagues. It doesn't really matter, you just need their names.

Goal: Write the six names as quickly as possible.

Materials: A pen and a timer

Use the table below for writing the names in Round 1 and Round 2 following the instructions.

Name	Round 1	Round 2
1.		
2.		
3.		
4.		
5.		
6.		
TIME:		

Round 1:

In this round you will be writing the six names one letter at a time.

1. Start the timer.
2. Write the first letter of the first name.
3. Write the first letter of the second name.
4. Continue writing the first letter for each of the remaining four names in its respective row in the table.
5. Now go back to the first name and add the second letter of the name.
6. Continue to the second name and add the second letter, moving to the third, fourth, fifth and sixth names.
7. Repeat for the remaining letters in each of the names.
8. When all names are complete, stop the timer and note the time.

Round 2:

In this round you will be writing the six names one at a time.

1. Start the timer.
2. Write the first name.
3. Write the second name.
4. Continue to write the other four names.
5. Stop the timer, note the time.

Compare your time from Round 1 with the time from Round 2. How are they different? Why do you think that is?

Commentary:

You probably found that writing the six names in Round 1 was much more complicated and difficult than in Round 2. It is also likely that it took more time. Although the effort of writing individual letters was exactly the same, the constant task-switching between names in Round 1 slowed you down significantly. With this simple exercise, it was easy to observe the effect of task-switching. The same happens when we work on multiple projects, switching between them as we go. Organisations often ignore the effect task-switching has on a number of factors important in software development such as cycle time and quality.

4.2. The down side: part one

From our earlier example, by delaying the second two projects we have helped the flow and therefore reduced average cycle time – but there is a major disadvantage. We assumed that there was no risk to the second two projects because we were still delivering them within a year. This is rarely the case. More normal is that when we serve customers sequentially, we risk losing the later customers or not capturing the opportunity. If we reject work – whether it is telling a customer we can't serve them right now or not allowing an idea to enter the development process in order to limit WIP – we lose valuable demand.

"Just say no", was a major advertising campaign in the 1980s as part of the war on drugs. It wasn't terribly successful in helping teenagers reject drugs, and it's not especially good at helping over-stretched IT departments turn down requests from customers or from the CEO. Most of us find it very difficult to limit our work in progress in this way.

It feels extremely risky. Perhaps the project is something already promised, or perhaps the project already exists, and effort has already been invested in it. There is likely to be internal resistance that is very hard to overcome: perhaps the CEO waves his arms about and shouts at the prospect of losing revenue he was counting on; perhaps the sales manager swallows his own tongue remembering how he promised a customer that the team could deliver... So what happens?

No-one has the courage to say no to the new project. Instead it is taken on but is under-resourced, gradually getting later, taking up space at the bottom of every report, annoying any customers who are waiting for the results and shuffled between managers like the hot potato no-one wants to be left holding.

Only a clear view of the overall value can really help us make difficult decisions about whether to take on something new, continue with an existing project or kill it. Constraining work in progress is a tool to help with overall value flow. If we are going to permanently reject potential demand, then we need to know that we are still raising the value we deliver overall.

Reassuringly, the idea that focusing on just a few things raises value usually turns out to be true. The extra value generated by launching a product earlier, normally outweighs the value from ideas which we could have pursued but which are now obsolete; serving one customer well (and getting paid), is usually preferable to risking losing two customers by being unable to complete the work in a timely manner.

"Stop starting, start finishing" was a pithy maxim that emerged from the Kanban community in relation to WIP limits. As a rule of thumb, it is better to focus attention on finishing a project, allowing you to realise its value, than to start on something else. The new project may have a large estimated value, but it remains speculative. Or to return to an older proverb: a bird in the hand is worth two in the bush.

CASE STUDY: Ticketmaster

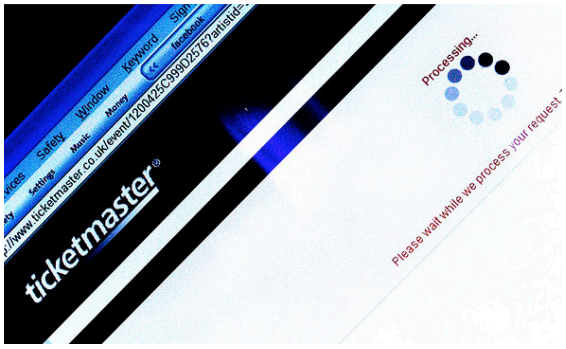


Figure 10. Trying to book tickets with Ticketmaster

When Michael Jackson announced his tour to the O2 Arena in London, Ticketmaster knew that demand would be high. Even they, however, had not anticipated quite how many fans would flood the site when ticket sales were announced. In order to manage the problem, they temporarily blocked demand to large numbers of fans in order to keep the website up and servicing customers.

This meant a risk – would fans leave the site and try to buy tickets elsewhere (perhaps from touts)? Nonetheless, since the alternative was a crash in which no one could buy tickets, Ticketmaster accepted a lower throughput than was ideal, in order to maintain flow.

4.3. The down side: part two

The other major disadvantage to rejecting work is that we lower our capacity utilisation. This feels very uncomfortable for teams – most of us are trained to feel guilty if we are sitting around not doing anything. 'I'm frantically busy', colleagues mutter, 'lucky you!' The subtext to such a statement feels like 'you are lazy'. Naturally, because you are in the lucky situation of taking this course, you will not suffer a guilty reaction. Instead you can snap out: 'actually, by decreasing my capacity utilisation I am helping the team improve cycle time'. After the colleague has given you a funny look and walked off muttering to themselves, you can sit down and consider whether to 'pull' in some more work in order to have something to do other than provoking your colleagues.

CASE STUDY: RPS Project, Swedish National Policy Authority

Every police car in Sweden is equipped with a small laptop, a mobile internet connection and a web application that allows officers to deal with minor offences there and then. There's no need to take notes, head back to the station, fill in forms and type them up – all the interrogation and follow-up requests are handled in the field. Obviously, such a vision required significant development to enable a user-friendly system to integrate with numerous legacy systems as well as complying with legislation and concerns regarding security.

The IT project was handled using a Lean approach which included WIP limits. As we've discussed, one of the issues of WIP limits can mean that occasionally individuals are left feeling they don't have much to do. In the RPS project, the team instituted measures to actively encourage those left stranded by a bottleneck downstream to help clear the bottleneck. For example, if system tests were causing a bottleneck in test, then developers would be urged to help out with manual testing and bug fixing, but also to help develop automated tests and improve the test infrastructure. The board was headed by a 'Question of the Week', in the case of our example it would be: 'How can we best help System Test today?' Rather than starting any new work, the team focused on finishing off features, clearing the bottleneck and attempting to improve flow to make a bottleneck less likely to occur in the future.

While it is true that maximum throughput rarely comes at 100% utilisation, if capacity utilisation goes too low, then our throughput will suffer. We know that lots and lots of cars will cause a traffic jam that makes everyone slower, but we wouldn't go to the opposite extreme of having empty roads to allow a very small number of cars to drive extremely fast. If your company needs the project to go through at maximum speed, then it is probably prepared to keep WIP very low and pay the price in efficiency. This is a pretty rare situation, however. Normally, we need to balance throughput, speed and capacity in order to maximise value.

Activity 6: What did you observe?

This activity should be run 3 times during the 6-week period that you are observing your workflow (see Activity 4). It should take no more than 15 minutes each time.

To run the activity, gather the people that you have involved in the previous activities and discuss the following:

- Have seen situations arise where your "To Do" queue has increased in size faster than the "In Progress" queue? Why you think it happened?
- Have you seen situations arise where people within the team have not been able to start a new piece of work because the WIP limit has been reached? What happened in those circumstances?
- At any point during the observation, were you tempted to adjust the WIP limit? How would you have changed it?

Commentary:

Introducing WIP limits comes with the benefit of reduced cycle time but as we've suggested, it isn't without its downsides. If your "To Do" queue is growing faster than the "In Progress" queue, you are probably observing demand being blocked. You are actively choosing to defer working on the items.

There is another scenario that you have probably observed: when Work In Progress limits are introduced people may become less busy, perhaps even to the extent that they are actively looking for work to do. In disciplined teams however, you see people applying themselves to work that is in progress so as to complete it sooner (thus reducing cycle time).

If this is the last of the 3 sessions that you are running, we'd suggest that you spend a bit more time reflecting on how your answers changed over time. Comment amongst the group on how the introduction of WIP limits has affected your team over time.

5 WIP CONTROLS

5.1. Temporary rejection of work

A permanent rejection of demand is an effective but crude form of WIP control carrying with it the risk that once our queue clears, we shall find ourselves with excess capacity and no valuable demand. Far better if we could halt the inputs to our work in progress and, once flow improves, pull more in.

Hang on! Holding work? That sounds like a queue in a thin disguise. Those were bad, weren't they?

Yes, holding work is simply placing it into a queue somewhere else, but different queues cost more at different times. Aeroplanes are a good example of this. If there is congestion at an airport, arriving planes are asked to fly in circles. This is a very expensive thing to do in terms of fuel and potentially risky as well. It's better to call a pilot as the plane is en route and tell him to slow down so the aeroplane arrives a little later. But best of all is to contact the aeroplane before it even takes off and delay it on the ground. The expensive queue (landing) is being limited by a less expensive queue (departure).

Software development can easily operate using the same system – we hold work in a 'ready' queue. This might mean a series of innovation projects held before they enter the active design phase. By working on three at a time rather than six, we will finish three projects sooner and have them ready for launch (with all the benefits that implies) – before starting on the following three. Of course, value sequencing still applies: the most valuable project with best chance of success enters the pipeline first.

The biggest queue that you are likely to find in software development teams using agile methods is the queue known as the project backlog. When a team 'pulls' in five user stories to complete during the sprint, you could argue that they are putting all the other user stories on hold. The job of grooming or ordering the backlog to ensure the most valuable/important features are being worked on first is simply 'sequencing the queue' by another name.

5.2. Limiting tasks within the process

A limit to WIP is as simple as capping how many tasks any one person or process can work on at a given time.

How do we decide what this cap should be? How does a team know whether they will get through five new features in a single iteration or only four? If we take on too few then we will lower our capacity utilisation and hence our throughput! Too many then we risk queues and longer cycle times!

As David Anderson writes:

> 'Do not fret over a decision to establish a WIP limit... Choose something! Choose to make progress with imperfect information and then observe and adjust.'

Just as with changing batch size, WIP constraints are reversible. We can always add more WIP. WIP limits are rarely applied so strictly that if developers are sitting around yawning with boredom, and building models of the Eiffel tower out of matchsticks, we refuse to allow them to work on another user story...

In a burst of over-confidence, we decide we can now complete ten user stories in a week. Alas, we have pushed things too far – queues begin to form and a bottleneck arises. Now, we need to attack the emerging queues and begin to pull in slightly more modest amounts of WIP. It's not the end of the world, because we see the impact of the changes instantly.

Don't forget that some spare capacity (slack) is necessary if we are to optimise flow – and that we can only judge how things are going if we measure and make visible our WIP and the queues.

5.3. Throttling

As we approach our WIP limits, we can be more and more demanding about how valuable a project needs to be to make it in. This means we shouldn't be in a position where there is no spare capacity should an even more valuable project suddenly arrive.

If you are serving a customer, rather than producing a new product, the same principle holds true through pricing. As you approach your capacity constraints, your pricing can increase. Airlines like Easyjet do this – as the plane fills up, the remaining seats become more expensive. Perhaps the most ambitious project in recent years was to apply the idea to all motorists within London. The congestion charge is applied during 'peak' hours with the intention of decreasing the number of vehicles entering the city centre. There is no congestion charge before 7:00 in the morning or after 7:00 in the evening, or on weekends.

The same principle works well in IT. Just as you might be prepared to pay extra for immediate delivery if you are doing last-minute present-buying and want to guarantee the arrival, so a customer can make the decision about whether to pay a higher price in order to guarantee a certain date or speed. This approach is known as 'classes of service', and it is used specifically to throttle demand as we near our capacity limits by passing the decision back to the customer. Indeed, this works just as effectively for IT departments working for internal users.

A common problem is that departments demanding priority or 'extreme hurry' on their particular pieces of work don't feel the pain of such requests. Applying an internal higher charge to the cost of faster service when IT is reaching capacity can be an effective way of ensuring that 'urgent' really means urgent, by passing the consequence of demand back to whoever initiated it.

David Anderson calls it an 'expedite class of service' and points out that you might choose to treat a project as a special case not only because of a more valuable demand but also to protect other types of value. If a product has been launched with a critical defect, the team might need to focus solely on fixing it. Expedite service would enable you to bypass queues and the usual work planning process in order to respond to an unusual opportunity or threat.

5.4. Purging projects

If we only allow higher value projects to enter our workflow as we approach our capacity limits, it follows that at the same time we should also consider purging low-value work. If a product has been in the development cycle for a long time, it is entirely likely that its value to us has decreased. It may have been held up in a queue, or it may have been continually de-prioritised as higher value projects came along.

We discussed how difficult it can be to turn work down, but companies are also reluctant to cut projects half way through. Thinking about all the money and effort that has been invested already, they refuse either to kill the project or to invest the resource required to finish it. Don Reinertsen calls these the 'zombie projects'.

Money and time that have been invested already is a sunk cost. You cannot recover this. Projected value should be measured against the cost to finish, not against the total cost. Actually this often works in favour of the zombie project: if only a small amount of extra investment is required to finish the project then even a low value project may well justify being finished. Finish or purge – one or the other.

CASE STUDY: Cutting the tail

Unilever – one of the behemoths of the Fast Moving Consumer Goods world, decided to focus its resources by cutting the tail. By this they meant purging the smaller brands that contributed little in the way of value. From manufacturing, managing, selling and supporting 1,600 brands that represented 100% of their value, they cut the number in their portfolio to only the 400 brands that brought in 92% of revenue.

Naturally, the rationale for doing this was that having fewer brands would mean the teams could cut overheads and focus resources to enable these 400 brands to grow even bigger – and the evidence suggests that this has happened. Unilever's super-brands – OMO, Dove, Magnum, etc. – have grown market share and in overall revenue through expansion into new categories. Moreover, Unilever's share price overall grew in the ten years following the exercise.

For Unilever, the rationale was not about cycle time, but about purging low value projects to focus resource on high value ones.

5.5. Shedding scope

As we realise that tasks within a project are taking longer than expected or that problems are proving intractable, a way to control the work in progress and therefore the delivery time is to shed requirements. Unlike purging projects, this is actually a fairly common technique. The team begin by dropping the most troublesome, lowest value features – the ones that they do not believe will make the difference between a customer buying or not buying.

Product backlogs within Agile methodologies make this easier, by emphasising the order of requirements. When time is the constraint, the product can be launched once the most valuable requirements have been delivered. Dean Leffingwell in his book Agile Software Requirements, writes:

> '...in the agile battle of date versus scope, the date wins... we'll fix two things, schedule and resources, and we'll float the remainder, scope (requirements).'

It does not, after all, have to be forever. If we think that a whizz-bang effect is going to delight our customers, we can always release it as an upgrade once we've tested the market with the basic product.

The problem is that IT has been criticised in the past for 'reducing scope'. Maybe you have even been on the receiving end of this complaint! If so there is no better answer than to quote Brooks, author of The Mythical Man Month:

> 'Trim the task... Where the secondary costs of delay are very high, this is the only feasible action. The manager's only alternatives are to trim it formally and carefully, to reschedule, or to watch the task get silently trimmed by hasty design and incomplete testing.'

It helps if these questions have been considered in advance – a list of nice-to-haves rather than essentials can ensure the team already knows which parts can be easily dropped and thus plan accordingly. Marketing may ensure they don't widely publicise an element known to be on the 'at-risk' list; the architects will keep the nice-to-haves loosely coupled; manufacturing may use a speedy but expensive supplier for launch, moving to a cheaper, slower supplier as volumes increase...

Unfortunately, the rule of thumb that kicks in inside our heads when a project is late is that in order to get it done, we need to assign more people to the project. If this were intended to attack the queues or add temporary capacity to clear the existing WIP, this might be an excellent solution. Sadly, in our experience, it is more likely that the new people will be asked to work on the features not yet begun. In short, the project has more capacity, but also more WIP. The original team struggles on with the queues they are facing with the current WIP. New people start the later features which are likely to be lower value. Add on the fact that larger teams are harder to manage (with spiralling communication problems and integration issues), and you won't be surprised to see that the extra capacity increases delay. This is where Brooks's Law really comes from: adding manpower to a late software project makes it later.

The solution is to shed scope, justified by a proper business case that incorporates the cost of delay.

Activity 7: The survival kit

This activity should take about 20 minutes. You can play it with a team of up to 20 people. You will need the list below, pens and paper.

You are in a small bi-plane which has crashed somewhere in the Andean mountains. You believe you are approximately 100 miles from a town. The weather is about -10°C.

Objective:

Maximise your chance of survival, leaving the plane which is in a highly-exposed spot on a mountain ledge.

Remaining in the plane is not practicable and you will need both hands in order to climb down from the ledge, so you cannot carry anything. You have one rucksack between you which can hold 5 items.

The following items are carried in the plane:

- Blanket
- Compass
- Small axe
- Tins of food x 10 (each tin is 1 space)
- Wire
- Loaded pistol
- Rope
- Cigarette lighter (no lighter fuel)
- Piece of 4m x 4m canvas
- Flare gun
- Plastic water bottle
- Bottle of whiskey
- Map of the Andes
- Knife
- Ball of steel wool
- Newspaper
- Can of vegetable shortening
- Spare pair of boots

Play:

1. Make a list of which 5 items you will pick to take.
2. Discuss with a partner and agree a joint list.
3. Decide on your list of 5 as a group.

There are several ways of deciding on a list, depending on your objectives. It is not easy. You might think – but we want to take everything! The constraint forces you to choose items that will assist you towards a specific objective (survive in one place, make signals, etc). Once this objective is decided then you can begin to decide on the essentials for that constraint.

Commentary:

IT projects work in a similar way. Naturally everything is potentially useful or important – otherwise it wouldn't be in the backlog in the first place. If you are simply asking yourself what is a 'must-have' you are likely to end up with an enormous list. If you say, what must I complete in order to achieve X within 4 weeks, you can cut down the list for that increment.

5.6. Timing/phasing work

Most development departments work on a range of projects at any one time. These projects will stress different processes at different times, resulting in the build up of queues. By managing the overall work in progress, this can be monitored and fine-tuned. For example, some projects may require a huge amount of prototyping, but the programming is relatively standard. Others require little design innovation up front, but plenty of programming hours and therefore lots of testing. If we know what the queues are at each process, we can choose which projects to work on upstream in order to impact on the amount of work we are going to be flowing on to each area.

This can work within the team as well, where work is phased to deal with the expertise of individual team members – this looks at team members working not quite as generalists and not quite as specialists, but as a hybrid of the two depending upon the requirements of the work in progress. We will go on to examine this in more detail when we consider capacity in the Trade-offs session.

6 CONCLUSION

In constraining work in progress we have examined the solutions and tools that impact on demand. The reason for this is that controlling demand tends to be both quick and cheap. In the Trade-offs session we shall go on to consider the alternative – supply. This generally means increasing or adapting our capacity. While equally effective, and indeed with its own specific benefits, the response time can be longer.

Once again, consider a motorway. An accident has closed a lane, causing enormous tailbacks. In order to reduce congestion, the police and traffic controls have stopped more cars from joining the queue by a combination of signs, warnings and physical closure of access roads. These are all 'demand' changes.

Next they need to tow away the crashed cars, clear the road and reopen the lane. This will change the 'supply' of motorway space from two lanes back to three. Obviously, this is important and it will improve throughput, but it takes longer. In the meantime, it is better for motorists to be prevented from joining the motorway and to take a different, slightly slower road home rather than spending hours in unmoving traffic.

Now that you have completed this session, you will have learned:

How to identify and make visible WIP in software development

- The concept of a pull system and the use of visual signs to enable its use
- Task boards and other methods of visualising WIP and sharing information
- Common benefits and pitfalls resulting from trying to make WIP visible

How to monitor and manage WIP at a personal, team and project level

- How teams and individuals pull or commit to tasks
- Monitoring 'outlier' tasks and taking action to correct or purge

The principles of limiting WIP

- Temporary or permanent rejection of demand
- Why lower WIP leads to fewer queues and thus shorter cycle time
- Why queues have different costs of delay at differing points leading to the value of holding WIP in lower cost queues

The potential downside to temporary or permanent WIP limits

- Permanently losing demand or causing customer dissatisfaction
- Underused capacity which can lead to inefficiency and reduced throughput

Alternative methods of WIP control

- Throttling – raising the bar for value as we approach our capacity or WIP limits, most frequently applied as 'classes of service'
- Purging – removing lower value projects to make room for higher value projects (depending upon a correct assessment of potential value vs. required investment)
- Shedding scope – removing low value or lesser value features
- Phasing – scheduling work to take account of the differing process requirements within or between projects

BIBLIOGRAPHY

Allue, X., 2009. Visual Management For Teams. Visual Management Blog. [blog] 11 February, Available at: <http://www.xqa.com.ar/visualmanagement/2009/02/visual-management-for-agile-teams/>. [Accessed 9 January 2012].

Anderson, D., 2010. Kanban: Successful Evolutionary Change For Your Technology Business. Blue Hole Press.

Bailey, D., 2010. Work In Process (WIP) Limits, Policies, Etc.. Los Techies: Se Habla Code. [blog] 14 June, Available at: < http://www.lostechies.com/derickbailey/2010/06/14/work-in-process-wip-limits-policies-etc/>. [Accessed 9 January 2012].

Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change. 2nd Edition. Addison Wesley.

Brooks, F. (Jr.), 1995. The Mythical Man-Month: Essays On Software Engineering. 20th Anniversary Edition. Addison Wesley.

The Institute For Fiscal Studies, 2003. London's Congestion Charge. [online] Available at: <http://www.ifs.org.uk/publications/1779>. [Accessed 10 January 2012].

Larman, C., Vodde, B., 2008. Scaling Lean & Agile Development: Thinking And Organizational Tools For Large Scale Scrum. Addison-Wesley.

Leffingwell, D., Widrig, D., 2010. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison Wesley.

London Evening Standard, 2011. Cheap Fares To Beat Rush-Hour. [online] Available at: <http://www.thisislondon.co.uk/standard-mayor/article-24022972-cheap-fares-to-beat-rush-hour.do>. [Accessed 10 January 2012].

McCandless, D., 2010. Information Is Beautiful. Collins.

MIT Sloan Management Review, 2000. Cutting Your Losses: Extricating Your Organization When A Big Project Goes Awry. [online] Available at: <http://hbr.org/product/cutting-your-loses-extricating-your-organization-w/an/SMR045-PDF-ENG>. [Accessed 9 January 2012].

Reinertsen, D., 1997. Managing the Design Factory: A Product Developer's Toolkit. Free Press .

Reinertsen, D., 2009. The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas.

Shalloway, A., Beaver, G., Trott, J., 2009. Lean-Agile Software Development: Achieving Enterprise Agility. Addison-Wesley.

Wikipedia. Parkinson's Law. [online] Available at: <http://en.wikipedia.org/wiki/Parkinson's_law>. [Accessed 8 January 2012].

Womack, J., Jones, D., 2003. Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Free Press.

VFQ Value Flow Quality

valueflowquality.com

emergn.com

by emergn