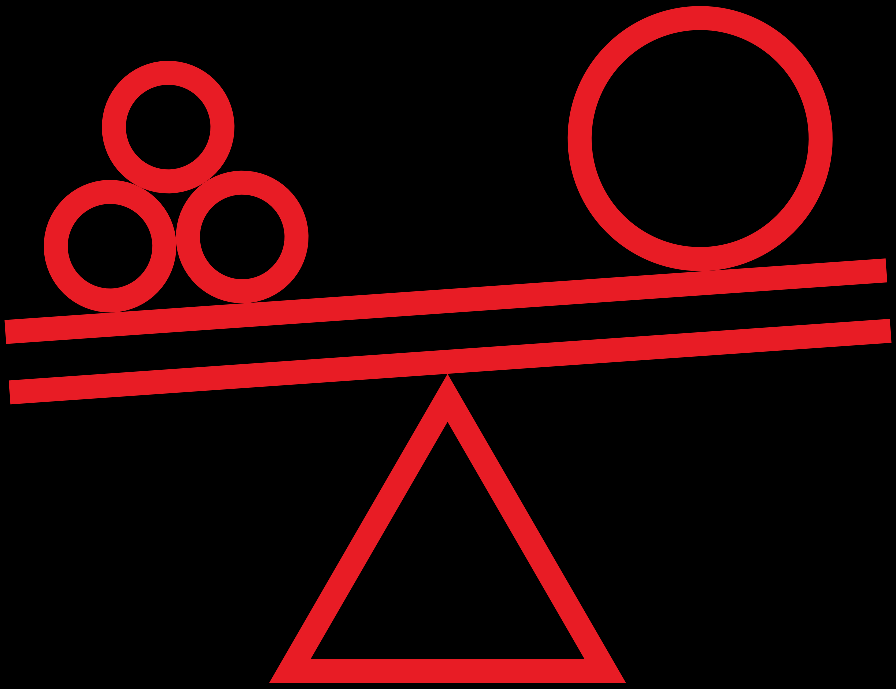
VFQ Value Flow Quality



Trade-offs

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.	Emergn Inc.
20 Harcourt Street	190 High St, Floor 4
Dublin, D02 H364	Boston, MA 02110
Ireland	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

Throughout the VFQ sessions we explore the tools that will help you optimise flow. In using them you will be able to decrease cycle time. While speed by itself is not the goal, a decrease in cycle time will almost certainly assist your organisation in delivering more value to your customers.

Decreasing cycle time also permits a shorter feedback loop, because we hear from our customer more quickly about what they like or don't like, what they are buying and what they are rejecting. This allows us to be more responsive, to cut short bad ideas and invest further in good ones. Because you can clear the pipeline speedily it means that your company can take advantage of an opportunity or change direction in the face of a threat. To learn more about the importance of feedback loops, read the Discovering Quality session.

The tools we have given you can be used in different ways and different combinations. Rather than providing you with a recipe book for how to use them, we want this session to make clear the rationale behind how you combine the tools to optimise a specific benefit. This is what allows you to create the recipe that's right for your organisation's needs.

The exact combination depends upon your business need and the customer's value. Because of that business need, there may well be one particular type of flow that will provide the most benefit. In this session we shall look at a model which examines how types of flow are connected and the choices, occasional trade-offs and differing emphasis that may be required to make them work for you.

We shall begin with a very quick recap of the rationale behind decreasing cycle time, before an equally quick reminder of the tools that can assist you. We shall then go on to look at where most companies sit now in their approach to cycle time and what is required to institute change. This leads on to examining the different focus you may bring to bear on a specific benefit from improved flow. We will present three main benefits or choices – we use the word 'choice' advisedly, since they are not mutually exclusive. A runner may need both explosive power and endurance. He is likely to focus on one for a specific race, but it doesn't mean that he has chosen to ignore or reject the other.

To optimise for each specific benefit within flow, you are likely to need a particular mix of your tools. This is not prescriptive, there is no absolute formula – but it is the mix we would expect to see used. It should be regarded as a starting point, to be adjusted and tweaked depending on what benefits your organisation wants to gain from flow and the effect the tools have as you begin to implement their use.

By the end of this session you will be able to:

1. Appreciate the importance of decreasing cycle time.
2. Comprehend a model regarding flow choices.
3. Decide where your organisation currently sits within the model.
4. See the attraction, and pitfalls of 'certainty' in delivery.
5. Appreciate the value of throughput and the likely combination of tools which will optimise for it.
6. Appreciate the value of flexibility and the likely combination of tools which will optimise for it.
7. Appreciate the value of all out speed and the likely combination of tools which will optimise for it.
8. Consider when certainty might be the correct choice and the likely combination of tools which will optimise for it.

1 DECREASING CYCLE TIME

Speed is not the goal, we told you at the beginning of our work on flow.

Have you watched Marilyn Monroe in Gentlemen Prefer Blondes? She is accused of being a gold-digger, wanting to marry a man just because he's wealthy and responds 'Don't you know that a man being rich is like a girl being pretty? You wouldn't marry a girl just because she's pretty, but my goodness, doesn't it help?'

Figure 1. Marilyn Monroe being admired by a crowd of men

Decreasing cycle time works rather the same way – it may not be everything to your business, but my goodness, doesn't it help?

Through delivering products to market faster, your organisation usually makes improvements in productivity (bearing in mind all our caveats regarding capacity utilisation). It also usually enables you to release value faster – both for customers and your company, which is great for staff, cash-flow and investors. It helps you gain market share and awareness, creating a positive cycle. Through a shorter, faster feedback loop, decreasing cycle time also helps you be more responsive to customers, to cut short bad ideas and invest further in good ones. Because you can clear the pipeline speedily it means that your company can take advantage of an opportunity or change direction in the face of a threat.

These are real, achievable results and by reducing cycle time your company is likely to enjoy them. Indeed it is these promises – increased throughput, greater flexibility and faster delivery – that lie behind the rush of so many companies to embrace methodologies which offer to fulfil them through a decreased cycle time.

We argue that unless you understand how decreased cycle time is brought about and buy-in to the rationale behind the methodologies, they often give disappointing results. This is because all of these methodologies require companies to embrace something else – something companies are often not ready to try and which every internal system militates against… uncertainty.

We'll go on to explore this in a bit more detail shortly, but first let's quickly recap the tools that assist in decreasing cycle time.

1.1. Tools for faster delivery

Faster delivery requires paying attention to your end to end flow. The enemy of flow is delay, and delay is most often seen, not in lazy developers failing to code swiftly enough, but in queues. These queues exist throughout our organisation, from approval processes before development begins to over-stretched specialists within the key development phase itself. By paying attention only to queues and things blocking us within development, we miss some of the easiest wins in reducing cycle time – in an area Reinertsen calls 'the fuzzy front end'.

Managing queues requires knowing that they exist. Making your work in progress (WIP) visible is one of the most important tools to do this. Constraining WIP then stops queues from forming at crucial points within the process, by ensuring that work is 'pulled' only when capacity is available. Where queues do form, when they are visible, ensuring you have an element of spare capacity enables you to attack and eliminate them immediately.

Batch size reduction helps us reduce cycle time, accelerate feedback and thus reduce risk. Since few software development departments are currently aware of their batch sizes (which are normally very large), it is an area of easy opportunity.

Finally, overlapping the different activities – analysis, development and testing, for example – by working on several phases simultaneously is a powerful way to speed cycle time. Many agile methods are based around this approach, ensuring that each team contains the entire skillset it needs to work on any batch, and carefully managing the team's external dependencies to minimise queues around approval, feedback and any specialist knowledge that may be required.

2 THE FASTER DELIVERY MODEL

At last – the moment every textbook dreams of – time for a graph!

But before we revel in axes, frontiers, segments optimal points and spheres of influence… a few health warnings.

No model – not even ours – is a perfect depiction of the world. This model is not intended to be rigid, nor to suggest that optimisation occurs at a specific point nor that there are hard divisions between each 'choice' or focus which we describe.

The focus which is right for your business and the resulting choices that you will make depends on the business need and the customer's view of time. Since you identified these in the Optimising Flow session, you should have a head start as you read through.

Let's start with the extreme ends of the graph – places where you don't want to be.

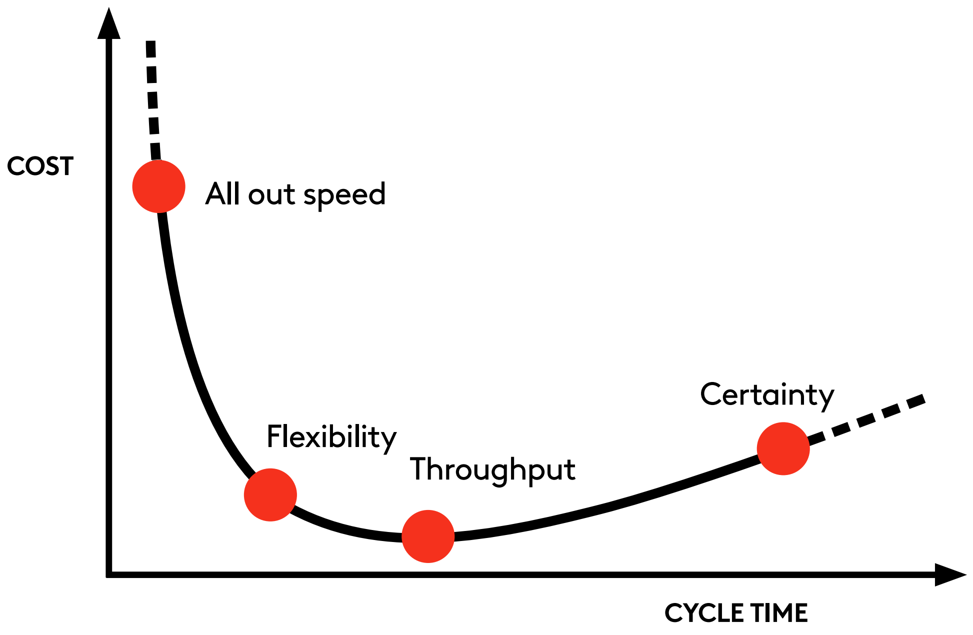


Figure 2. The Faster Delivery Model

2.1. The extreme top left: where extra cost leads to no appreciable gains

If we want to get an individual from A to B, one Ferrari will get her there fast and cost quite a lot of money. Buying three Ferraris will not get her there any faster, they will simply cost three times as much. Hiring a private jet may make the portion of travel faster, but the time spent organising a take-off and landing slot may mean there is no appreciable increase in end to end speed.

2.2. The extreme right: where extra time means costs begin to creep up

In trying to get our friend from A to B, the company has decided not to hire a Ferrari, or indeed any transport at all. Since they want to spend no money, they have asked her to hitchhike, and where that fails, to walk. We can be fairly certain she will get to B eventually, but she may arrive too late for whatever purpose we wanted her at B for in the first place. Worse, eventually costs will start to creep up. We may not have hired a vehicle, but the company is still paying our friend for her time – the slower her progress, the more days they have to pay (let alone the cost of bed, beer and breakfast at a wayside inn).

2.3. What happens as cycle time decreases?

As a company travels from the far right towards the left of the graph it typically sees improvements in every area (just like all the training courses, books, coaches and other paraphernalia promise). As you can see, cycle time is decreasing, cost stays low for a long time or may even decrease, while shorter cycle times improve flexibility. But past a certain point choosing on which benefit to focus becomes more important.

Figure 3. Couch potato!

This is rather like someone – who eats too much and never leaves the couch – going to the gym. At the beginning everything is going to get better – they will lose weight, get fitter, become more flexible, increase strength... After a while (a long while and assuming they haven't given up after 2 sessions because it's difficult), they will need to make choices within their training regime based on what they want. Are they interested in being slim? In that case lots of cardiovascular work and a limited amount of toning weights exercises will be better. Are they keen to excel at a certain sport? That will require a tailored fitness plan depending on the sport. Do they want to improve a sense of wellbeing? Yoga and Pilates might prove more effective than boxing.

3 WHERE ARE WE NOW?

We cruelly compared a company on the right of the curve to someone unfit and overweight. You must have heard people referring to a project as 'bloated', 'top-heavy'… we don't think the use of such words is altogether coincidental. The metaphor works because where most companies choose to put themselves by default is in a flow choice that involves quite inordinate amounts of buffers – padding for budgets and schedules. And what does padding look like? Yup. It's not a pretty picture.

Why would we choose to pad our schedules and our budgets? It makes us slow and expensive and those are two things most companies are keen to avoid. Few board members have stood up in a meeting, slammed their fist on the table and announced 'we need to go slower and we need to raise our costs.'

So why are we here? The answer of course is that we're not choosing to be slow and more expensive than we need be. We're choosing something else – something we think is terribly important – certainty.

Activity 1: Certainty of your projects

This activity should take about 15 minutes. You can do it on your own or invite a few colleagues to join you.

In this activity you will explore the projects that are currently being developed or have recently completed in your organisation.

Preparation:

- A stack of index cards or similar.
- Pens.

For every active or recently completed project or a piece of work, you can think of, create a single index card and write the project's name at the top. Include large, long-running projects as well as small changes that only took a few days.

Now pick the cards you have created one by one and write on the left a list of all on-time metrics that you know the project has defined. Typically you may find a fixed delivery date or an expectation of how long exactly the project will take; list any deadlines or milestones referred. If there are none, leave the space blank.

Next, go through all the cards and try to list on the right side of the card any potential reasons why the project required the on-time delivery, what dictated the deadline selected, how was the delivery date established. If the piece of work had no on-time measures, list why there didn't need to be any.

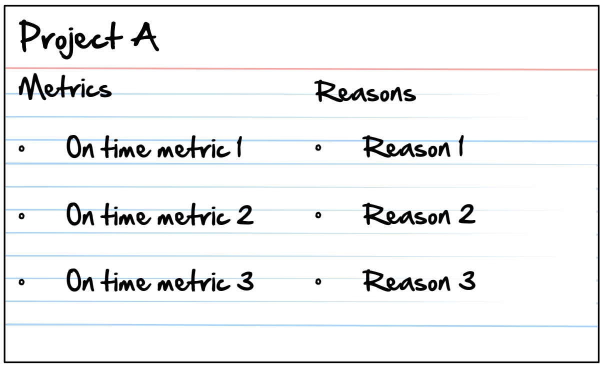


Figure 4. A sample index card

Now arrange all the cards on the table and group them together looking at the left column – the projects and pieces of work with explicit time-related expectations against the ones where none were defined.

How did the split work out? How many projects did you find with time-related measures? How many of your projects lacked them. Looking more carefully at the projects with a time-related measures explore the right column. How relevant are the reasons listed? Are they customer focused or internally focused?

Commentary:

Time and time again we find that organisations expect certainty of delivery for their projects by specifying strict deadlines. In many cases these are completely artificial and have no ties back to the real customer needs.

Emergn View

Emergn would argue that almost 90% of companies remain on the right hand side of the graph, delivering work against a "certain" delivery date and budget. Although costs are closely controlled, cycle times are long – indeed so long that these companies are often paying more than the optimum.

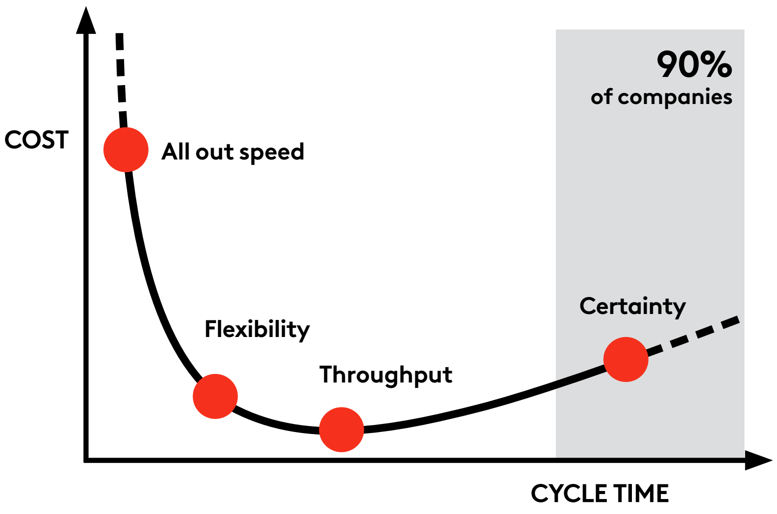


Figure 5. 90% of companies deliver work against a "certain" delivery date

When we use phrases like 'set in stone', 'drop dead date', even 'deadline' itself, we are signalling how rigid our approach to these schedules is. We are judged by this schedule, measured against it, perhaps even paid according to it. We begin to build dependencies around it – advertising space, shelf space, launch parties, press announcements. Perhaps our customer has built all these expensive dependencies into their launch date so we had better deliver the software that makes their product work… Guarantees are asked for and given. To meet them we build enormous buffers and amounts of padding into our schedules.

There are very important reasons why we worry about deadlines. We aren't advocating telling all other departments to chill out and go with the flow… but we are asking companies to look at what 'certainty' really means, the cost that we pay for it, and what benefits might occur if we dare to give it up.

3.1. What is certain anyway?

"In this world nothing can be said to be certain, except death and taxes."

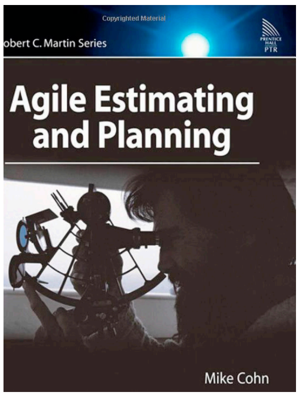
Benjamin Franklin

Strictly speaking, certain delivery ought to mean a requirement to hit a certain launch window, based on a customer need – Christmas products needed in December, exam results which must be delivered in August, etc. Since this window is incredibly important and the value of the product will go to zero if late, the company needs to build in plenty of buffers to ensure that they can hit it.

When it comes to Christmas deliveries, Royal Mail provides a list of 'last posting dates' by which you need to post your cards to be sure of them arriving on time. If you want to send your cards to the Far or Middle East by surface mail, the final date is 30th September. That's a full 86 days before Christmas.

Royal Mail is trying to reach a point at which the risk of not meeting a Christmas delivery is so small that they can discount it. The reality is that most post will arrive well in advance of Christmas. Unlike Royal Mail, however, companies tend to use their contingency time. This is as if the postman arrived and decided that since he was in Jordan he might as well take a trip to see Petra. Padded schedules often lead to the slowest delivery. Because a benefit of 'certain' delivery is how easily other departments can plan carefully around the schedule, it also ensures a high level of dependencies. This in turn makes it very difficult to alter course or to deliver earlier, because any change will cause disruption to the schedules of numerous other departments.

"We ignore uncertainty about the product and assume that the initial requirements analysis led to a complete and perfect specification of the product. We assume that users will not change their minds, refine their opinions, or come up with new needs during the period covered by the plan. Similarly, we ignore uncertainty about how we will build the product and pretend we can assign precise estimates ("two weeks") to imprecise work. We cannot hope to identify every activity that will be needed in the course of a project. Yet we often fail to acknowledge this in the plans we create."



Cohn, M., 2006. Agile Estimating and Planning.

We included the above quote at length because we wanted to emphasise that uncertainty is not actually as frightening as it might sound. Traditional planning and scheduling which commits to a certain date and builds in endless buffers to protect it is still not an actual guarantee: accidents do happen. In the example we gave about the extreme right of the graph, where a company ask an employee to walk, we said 'we can be fairly certain she will arrive eventually'. Fairly certain, but not absolutely certain: she could get lost, she might fall over and twist her ankle, she might get hit by a car...

In software development projects anything might derail the project, from frequent occurrences such as the business changing its strategy or the project proving more complex than expected to the truly unusual – an event like 9/11 in New York which impacts capital expenditure.

Perhaps what we should say is that change requires us to give up the illusion of certainty. Funnily enough, people seem to find illusion almost as painful to sacrifice as reality.

3.2. Exposing the illusion

The traditional method for dealing with uncertainty is to introduce large buffers. These are in effect a form of queue that pads each activity. There are two major problems – the first is the problem we have pointed out before – contingency tends to get used, filled up with small, low-value features that imperil the critical path activities. The second is that inherently uncertain tasks or projects – which is almost always the case with IT – require more contingency.

As we've pointed out, you can never reduce risk to zero, only to a level with which you are comfortable. In reality, instead of discussing what level of risk the company can manage for this project, a small contingency is added with no real justification behind it, the risk ignored and a commitment given to certainty. The opposite extreme – trying to analyse every contingency – leads to enormous batches. Detailed planning is done at the outset to create exact specifications for every possible requirement. Not only does this lead to lengthy cycle times (read the Batch Size Matters session if required), but it is unlikely to be accurate anyway. Far more likely it will turn out to be wasted work when the customer decides it's not what they wanted after all, your team discovers something has changed, the market has moved on or a competitor has launched something faster (even if their product is not as good as the brilliant idea you have sketched out on paper).

Uncertainty is at the heart of IT. It is a creative, knowledge-based discipline in which problems are continually changing and where most tasks are non-repetitive. Companies ignore this at their peril. It's not surprising that the Standish Group found in their Chaos report that:

> "On the success side, the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget."
>
> **The Standish Report**

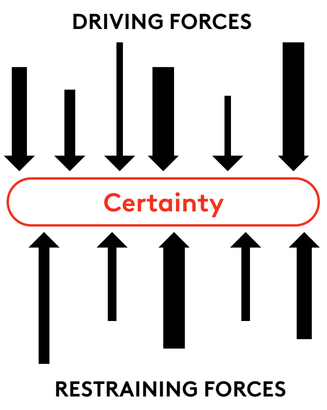
Instead of joining such disappointing statistics, most organisations would benefit enormously on reconsidering their default choice. By accepting uncertainty they can become more efficient, faster and more flexible – and those are the kinds of aims you do hear board members stating in meeting rooms.

Activity 2: Certainty in delivery

This activity should take about 10 minutes. You can do this on your own or with a few of your colleagues.

You have now read about certainty in delivery, and the different advantages and disadvantages of this particular flow choice. Reflect on the arguments presented in the text above and how they apply to your company. Add issues that may be specific to your situation and create a list of driving and restraining forces related to selecting certainty in delivery.

Now create a "force field analysis" diagram like the one presented below.



Start with the "Certainty" in the middle. Draw a labelled arrow pointing down for every supporting force and an arrow pointing up for a restraining force. Represent the strength of each force in your organisation by the thickness of each arrow.

What does your picture look like? What is the balance of the forces you have drawn? Which ones would be easier to influence, which are more challenging?

Commentary:

Often, when people undertake an examination of the motivation to get a high degree of certainty in software delivery, they uncover reasons that might suggest the need for an alternative approach. What did you uncover? Can you still justify "Certainty" as the right flow choice for your project?

4 WHERE MIGHT WE WANT TO GO?

When we embrace uncertainty and begin to implement the tools which can lead to faster delivery we will normally see gains in each of our desired areas. There is no guarantee of course, but funnily enough, because we are managing our flow better and exposing hidden queues we often actually increase a measure of predictability and therefore certainty. Hence why many especially passionate advocates of agile methodologies claim that there is nothing to give up at all. We would say that this is an advantageous side-effect of using the tools well – as soon as certainty becomes your stated aim, you pay in terms of cycle time and flexibility.

Now let's look at what happens when we focus on a specific benefit – a very popular one as it so happens, since it seeks to optimise productivity – getting the most amount done, for the least amount of money, in the shortest possible time.

4.1. Throughput

A lot of companies are very interested in maximising their throughput – so interested that we are going to cover the subject in more detail in our Getting More Done chapter. Here we will examine the relationship between our key development tools and how they should be best combined for this flow choice.

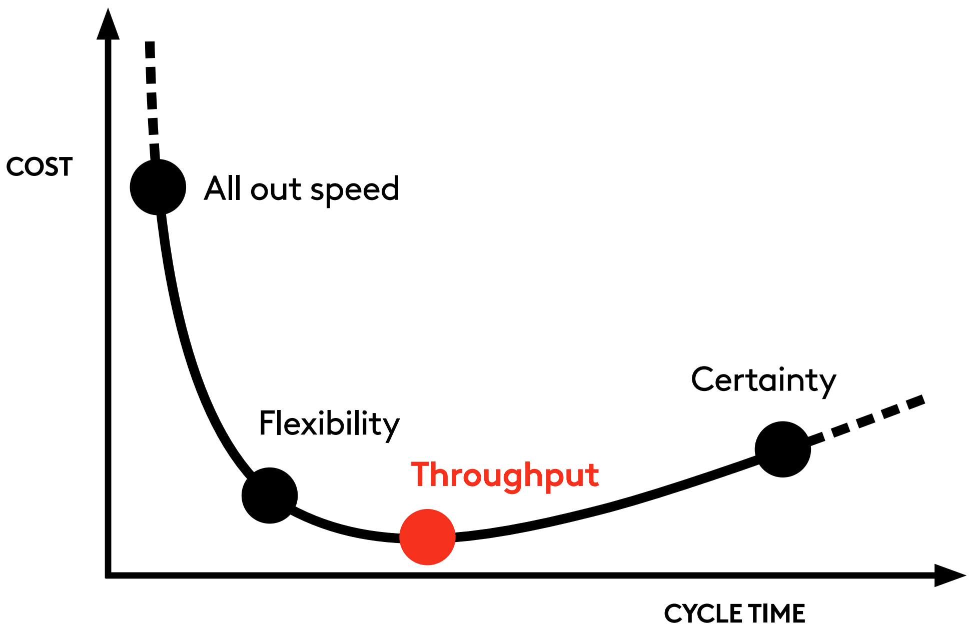


Figure 6. The Faster Delivery Model

Checking throughput is right

Henry Ford wanted to make motor cars more affordable. He believed that more people would buy cars if they were cheaper.

But how would you make a car cheaper? These were specialist items, made by craftsmen, which was the way top carriages had been made for centuries (after all why bother making a cheap carriage? Only the rich could afford to keep horses anyway).

Ford's revolutionary insight was to apply the kind of processes already in existence in factories manufacturing soap and chocolate to car-making.

He set up his factory using moving assembly lines, to be able to produce identical cars at high volume, raising productivity and lowering costs. This was throughput in action. By producing large numbers of cars within a smaller space of time, as well as by using cheaper materials, Ford was lowering the cost of each individual unit.

Figure 7. Model T Ford assembly line - from the collections of The Henry Ford and Ford Motor Company

This worked because of something very important – Ford was right in his belief that people wanted a cheaper car and that they would sacrifice bespoke variation to get it.

In software, where value does not come neatly packaged in cars, the idea of throughput is more frequently seen not as unit cost but as overall cost - a fixed budget within which developers try to produce as much as is possible.

Thinking about Henry Ford remains useful, because software developers need to keep an eye on why Ford's method worked. We need to know that what we produce is valuable. If our developers crank out 1,000 lines of code a day, managers might be delighted – how productive! But if this code turns out either not to work as intended, or not to be what the customer wanted - then the developers' super high productivity would be simply a waste of time and money.

In throughput, there are two key dangers. The first is that we build something perfectly but then discover the market doesn't want it. The second is that we build something wrongly and have to redo it. In both cases the reason for our failure is that we haven't received feedback sufficiently swiftly to be able to incorporate it.

If we wish to maximise the amount of work that we can do for a given amount of money (which can also be thought of as equivalent to a low per unit cost) then we need to think rather like Henry Ford. We need to know what the customer wants and we need to know how to build it. Preferably we want very low variation within the work itself as well (tasks of a standard length, difficulty and value). These conditions are not usual in software development, which means a focus on optimum productivity against cost is not the solution as often as budget-obsessed managers would like to assume. On the other hand if you have done something before and really do have solid feedback, then throughput may be the most responsible and profitable way to manage your flow.

Queues and capacity

Figure 8. Patients waiting at a GP's surgery

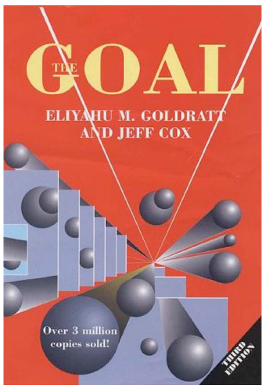
We discussed the way a GP's surgery worked and we mentioned that the throughput efficiency was optimised from the GP's point of view. Because of this any unexpected variation or delay could, and indeed often does, cause large queues. If one patient overruns by 5 minutes, all other patients will be delayed. Delay for patients is considered an acceptable state for the surgery. In a private health system where the patient's choice, opinion and waiting time is considered more important (that is, it has a higher value since the patient may walk away), the surgery will be run slightly differently in order to minimise the delay to patients – in practice this normally means running with spare capacity.

Maximum throughput does not occur at 100% capacity utilisation. An engineer in a traditional factory does not run the machines 24 hours a day. He knows that it is crucial not to build up inventory from one machine unless it can be processed all the way along the manufacturing line, and he also knows that maintenance is a crucial part of efficiency – a machine that breaks down will cause a bigger delay than scheduled halts for maintenance, for example.

Identical logic must be applied to the development process. We must consider end to end flow, maximising efficiency of the whole and not simply a single element. A specialist who is overloaded, for example, is a classic sign of efficiency at a single point which ends up translating to increased cycle times and therefore reduced throughput overall. The system needs to carry a reasonable quantity of spare capacity, and/or be able to tolerate a certain level of queues and the resultant delays. Of course, if you have read the Attacking Your Queues session, you will be aware that since long queues get longer and cause disproportionate damage, these queues must be measured and watched. Since increased cycle time is unlikely to benefit your value flow, your organisation must decide exactly what level of queue can be tolerated and then attack emergent queues beyond this decisively.

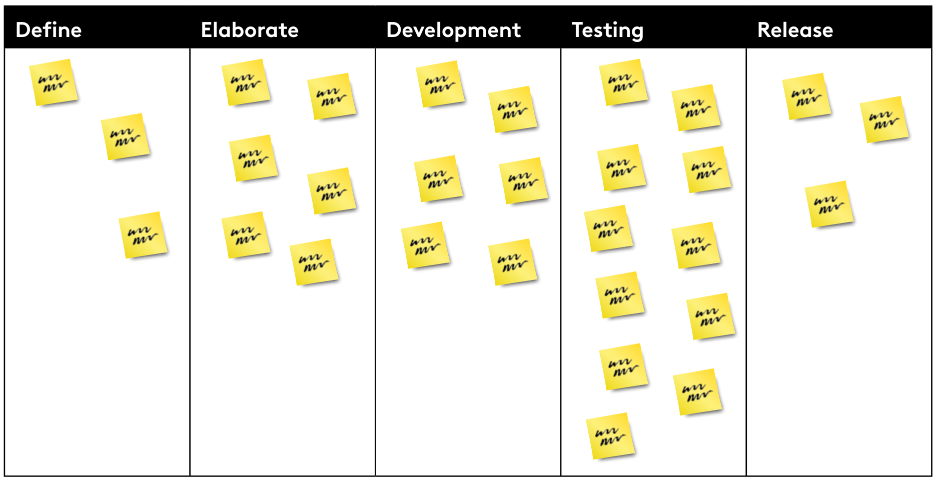
Activity 3: Should we alleviate the constraint?

This activity won't take any more than an hour to complete.



In a system whose goal is to maximise throughput in order to meet a given outcome, constraints that are likely to cause bottlenecks become a concern. Eliyahu Goldratt, author of the book The Goal (Gower Publishing Limited, 2004) introduced the Theory of Constraints as a philosophy that can be used to optimise flow within a manufacturing system. It proposes five focusing steps necessary in order to achieve this: Make the goal of the system explicit, identify the system's constraint; decide how to exploit the constraint; subordinate every decision to the constraint and finally, elevate the constraint (make changes elsewhere to break the constraint). In manufacturing the Theory of Constraints is useful to us as our constraint is likely to be a global one, which affects the whole system. In the case of software development, constraints emerge unexpectedly and often exist in more than one location at a time – they are far more dynamic in their nature than those in manufacturing.

In the Work In Progress session, the activities asked you to visualise your process. Visualisation is a really effective means to identify the constraints within a software development system.



If you haven't had the opportunity to do the activities in the Work In Progress session yet – now's the time to do so. If you have, great! Use the kanban board you created (and are of course continuing to use) to identify a constraint in your process. In this activity you will work out whether it would be worth elevating the constraint by calculating the cost of capacity at the constraint and comparing it to an approximation of the Cost of Delay (for a more comprehensive approach to its calculation, see the Technique Library.) for the work being undertaken.

First, approximate the Cost of Delay for your project; an easy way to do this is to revisit the business case that established the need for the project sum all of the claimed ROI numbers from there, total them and divide that number by 12 to give you the cost of a month of delay. Those of you who have done Activity 8 in the Optimising Flow session may find this part of the activity a little familiar as we're repeating what you did there; if you still have the number that you came up with in that activity you can save yourself some time, by using the number you arrived at previously.

For the next part of the activity you'll need to understand the Cost of Capacity at the constrained resource. One way of approaching this is to understand the annual costs of things like the people involved, any hardware needed to undertake the task, etc.

Use the table below to note whether the cost of the associated additional capacity in return for the associated reduction in delay would be a worthwhile investment.

Reduction in Delay	Cost of Capacity Necessary	Economically Sound
1 Week		
2 Weeks		
1 Month		
3 Months		

Commentary:

It's typical that when requests for additional people or resources are presented on a project, the costs are scrutinised heavily and a decision made on them alone. In this activity we present a way of approaching the decision in order to make it more subjective – a comparison of the saving received by avoiding delay, and the cost of capacity required to achieve that saving.

Variation and batches

We mentioned Henry Ford and his factory as a model of throughput. Henry Ford famously said ‘any colour as long as it’s black’. He knew that his particular style of manufacturing could not cope with variation or diversity.

Eventually, other manufacturers began to attack Ford in exactly these weak spots – keeping costs low enough to stay competitive, but adding innovation and variety that customers were prepared to pay for. We mentioned how often in software customers DO want variation. The ability to change and respond to variation, we have summarised as flexibility. Since organisations are used to a lack of flexibility in their development processes, they are unsure of how increased flexibility would benefit them.

There is a very simple test to know if we have chosen correctly: are we getting it right first time? If we release a small batch and it turns out to have zero defects, positive user feedback and happy customers then we know that we don’t need to speed up the feedback loop and we can concentrate on throughput. If we pick up a lot of bugs, our customers aren’t satisfied or are requesting lots of extra functionality or not using a feature we thought was crucial, then we know that we needed more flexibility in order to speed up feedback.

This test is very powerful, but you’ll notice it requires something important – releasing that first small batch! That’s why throughput should focus on creating a batch size that promotes the delivery of value early.

> “By carefully choosing the way in which software components are assembled, we can create identified units of marketable value well before an application is anywhere near completion.”
>
> **Deene, M., Cleland-Huang, J.,** 2004. Software by Numbers

This is referred to in different ways – we’ve already spoken about the ‘Minimum Viable Product’ in Lean Start Up, but you may also hear agile methods referring to a ‘minimum marketable feature’, Scrum talking about a ‘potentially shippable product’ and XP emphasising the need to deploy increments of value.

Note that this is not the same as the smallest possible batch size. While large batch sizes unquestionably cause delay and should be avoided, smaller batches potentially increase transaction cost. Bearing in mind all the benefits we urged in the Batch Size Matters session that count against worrying too much about transaction cost, there is still a time when a project may well be happy to increase batch sizes for reasons of efficiency.

As a project progresses, for example, you may choose to increase batch size, gaining feedback less frequently. Of course – if you find your error rate rising with feedback that suggests you have gone off course, you will want to reduce the batch size again.

WIP

If you have chosen throughput correctly, then it means you have more certainty about the value and contents of each batch. That permits you to allocate your resource with a higher certainty of getting it right – in practice, that means running with slightly higher capacity utilisation, which means you are also likely to run with a slightly higher level of WIP and the resulting queues. This makes the visibility of WIP even more important. Only by maintaining visibility of this at all times, can the organisation be sure that we have judged this correctly. If batches begin to arrive with high variation, our resource allocation and high capacity utilisation will start to work against us. Long queues will begin to form, and being able to see this and take the appropriate action is essential if we don't want to damage the project's value through long delays.

There is no exact recipe. We cannot say: limit your WIP to 5 items to maximise throughput. The right level for your project depends upon your own unique circumstances. What we can say is simple – experiment with your WIP limit until you find the best level to optimise flow.

This system works best when combined with the concept of a kanban pull system. The ability of teams to self-organise their own WIP, pulling new tasks as they have capacity, is much faster and more productive than waiting for work to be assigned by a project manager.

Development processes

We have discussed the importance of a measure of spare capacity above. A focus on throughput also benefits from the use of concurrent processing. However, as pointed out in our chapter on faster development processes, breaking down dependencies is not always easy and it can have associated costs. Whether your organisation is prepared to bear these or not depends on the value it expects to achieve from the extra speed. In general, we would expect that companies focused on throughput might tolerate a higher level of dependency – indeed, these dependencies might be where you find the key queues.

We would urge all organisations to include essential development elements such as testing and integration within a cross-functional team as part of releasing smaller batches. But for reasons of cost efficiency, other resources might remain shared. For example, your organisation might have a single prototyping department, which is a constrained resource – it will be up to senior management to sequence any queue for this specialist resource in order to ensure that the most valuable projects with the highest cost of delay are being serviced first.

CASE STUDY: The Microsoft XIT team

The XIT team at Microsoft had a truly horrible reputation for customer service. They were in charge of doing minor upgrades and fixing production bugs in about 80 applications used around the world by Microsoft staff. The team had an enormous backlog of 80 items at any one time and a 5-month lead time on new requests. Naturally, this meant that the backlog was growing – a classic example of a queue spiralling out of control. There was no extra money – indeed, managers had the impression of a badly-organised, failing team and were reluctant to hand any extra resource to the problem. It was clear that the priority had to be clearing this mass of work: getting more done at a faster speed.

The good news was that each item only took about 11 days of engineering. That meant most of the lead time was waste that could thus be eliminated. The manager in charge of the programme decided that part of the waste was the process of estimation. Estimating a job could take a third of the team's development time. On a bad month it could creep up to 40% because the company demanded that estimates be returned within 48 hours. Obviously because a batch of estimates took priority, and arrived randomly, this also had a negative effect on predictability.

The manager decided that the best way to increase throughput would be to eliminate some of the variation. To do this, he persuaded the product owners to stop requiring estimates. He calculated that since almost all estimates were of around the average time of 11 days, he could work with this figure, simply keeping an eye on any outliers.

This wasn't an easy change for the product owners. They used the estimates to assign costs within the budget. Instead the XIT team manager now asked them to pay a fixed proportion. Rather than assigning exact costs to each department based on the estimates, he would assign them a proportion of costs based on the average engineering time of 11 days. He also limited the amount of WIP – product owners could only pick their top three items and put them into development as and when there was a slot. The lack of flexibility sped this up – they knew their top three without any need to reprioritise the whole backlog every month.

Now the XIT team could simply reject any work that seemed too big (more than 15 days) as and when it was pulled in to development. They had gained nearly a third extra capacity by dropping the estimation process. Since they were empowered to reject large items, they were at no real risk of using up their capacity on the wrong sort of project. The product owners paid a fixed cost and received a guaranteed amount of capacity.

The throughput and productivity rose. The team agreed to deal with requests within 25 days rather than the previous 5 months – they were quickly able to hit this target. Indeed, within a year, the team had brought the average time down to 14 days and had eliminated the backlog. In fact the productivity of the team rose by 200% - not through extra capacity, but by limiting variation to remove the non-value adding activities.

Summary

Throughput relies on having a great deal of up-front knowledge – so much that we are confident we do not require much feedback. Because of this we can emphasise efficiency, running at higher capacity loads, allocating our resource exactly and allowing higher WIP and larger batches. A key change from where many companies sit now, is that instead of worrying about certainty of delivery, throughput provides good predictability as long as the team focuses on the early delivery of value – something which often involves spare capacity or other forms of investment. Remember – Henry Ford wanted his cars to be cheap and he wanted customers able to buy them quickly. He chose to invest heavily in automation in order to make his faster delivery a reality. It paid off – by 1914, Ford had a 48% share of the automobile market. The exact trade off between costs that assist faster delivery and the cost of delay must be worked out on a case by case basis to be sure that the balance has been achieved for your project.

4.2. Flexibility

There's a clue in the name: agile methodologies focus on what we see as a flow choice that emphasises flexibility or the ability to embrace change. While agile practitioners stress a multiplicity of benefits when adopting agile processes, we feel that most are either secondary to or stem from the true focus on flexibility.

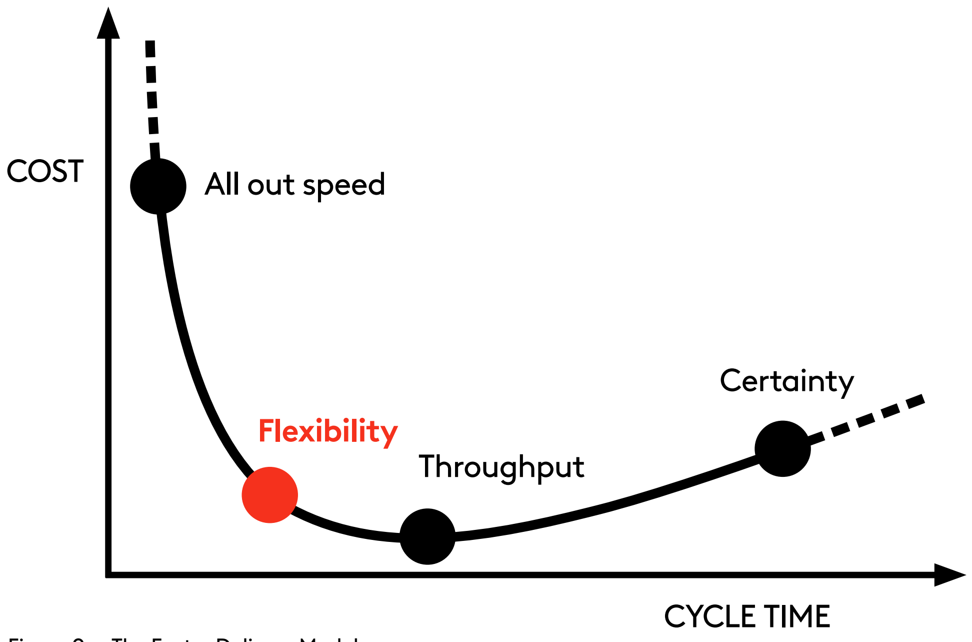


Figure 9. The Faster Delivery Model

The Agile Manifesto states as one of its principles:

PRINCIPLES BEHIND THE AGILE MANIFESTO

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Figure 10. Second principle of the Agile Manifesto

Preston Smith defines the utility of this approach in his book Flexible Product Development as "The ability to make changes in the product being developed or in how it is developed, even relatively late in development, without being too disruptive. The later one can make changes, the more flexible the process is. The less disruptive the changes are, the more flexible the process is."

You are redecorating a room in your house. On your day off, you decide that the cheapest and quickest way to do it is to dash out to the hardware store, buy a 50 litre can of white paint and set to work.

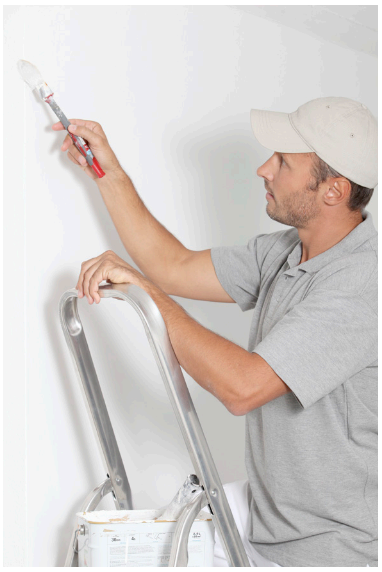


Figure 11. Painting the room white

When your spouse returns from work he or she loathes the colour as too sterile and boring. Loudly abusing you for your lack of taste, your spouse goes back to the hardware store and buys a can of deep red paint and you both repaint the room that weekend. When finished your spouse reluctantly agrees that the bedroom is now too dark. The next week you hire a designer who recommends leaving one wall in red as a feature and painting the rest white. When your spouse sees the interior designer's bill the trouble really starts...

At the outset you had considered buying lots of sample paints, putting them in patches on the wall and then seeing what you thought. Given that the sample pots were £5 each for just a few millilitres and that you were keen to get the job done on your day off, you had rejected this solution. With hindsight, you wish you'd done that and painted only one wall at a time, consulting with your spouse frequently.

Flexibility works by setting things up to keep choices open. It understands that what looks like an efficiency (huge tins of paint, doing all the work in one day) may eventually end up costing more and taking longer. Just as in our example, however, flexibility involves uncertainty (what colour are we going to end up with?), potentially more cost at the beginning (all those sample pots), and a sacrifice of potential efficiency (after all, if your spouse had loved the white walls, your solution would indeed have been right).

Choosing where and when to be flexible

Our example highlights an important point – change costs. The question is at what point change costs the most. Changes that occur at one point in the process may cost very little, at the next they may be disastrous. Changing your mind about which paint colour you want before you buy the large tin of paint is cheap; and changing your mind about the colour after painting one wall is cheap compared to changing your mind after you've painted the whole room.

This impacts heavily on software development because change is an obvious and visible cost, which means it is often resisted. As Preston Smith writes, "Managers resist change in a project – quite correctly – because it is expensive, and change usually leads to schedule slippage. Furthermore, change can open the door to product defects... In general, the cost of a change rises the later it occurs in the project."

It takes a certain amount of vision and bravery to know when to swallow the cost and make a change, and when to ship something even if it's not perfect in order to keep costs low and have more time in market. Some of the most famous examples in IT can be found under Steve Jobs's leadership at Apple – famous because he sometimes called it right, and sometimes wrong. When design on the iPhone was nearly complete, Jobs called a halt. The glass screen fitted neatly into an aluminium case, yet Jobs decided that it wasn't quite working – it was too masculine and it didn't showcase the 'hero' nature of the special glass screen. The team redesigned the phone – at an enormous cost. They had to redo the circuit boards, antenna and processor placement. This choice was almost certainly right – the iPhone did sell on its physical design and its revolutionary screen. But choosing to delay the Apple Macintosh launch in order to perfect the cube design of the casing was probably wrong. This design detail did not have sufficient intrinsic worth in a customer's eyes to justify the delay.

Figure 12. An Apple Macintosh computer from 1984

In general, it's commonly accepted that the cost of making a change rises over time. This is what we would expect – and companies normally charge those who want complete flexibility a premium to reflect this. Train and plane companies, for example, charge more for a ticket if you wish to be able to use it on any day, at any time and to be able to change your booking. This is because your last-minute changes may mean they have to run with an empty seat which they might otherwise have sold.

Advocates of agile point out that teams spend a great deal of effort in trying to minimise the cost of making changes. Kent Beck comments on how 'automated tests, the continual practice of improving the design, and the explicit social process all contribute to keep the cost of changes low.' This is true, but such effort does not negate the fact that the cost of making a change does exist, and does rise as time goes on. After all, eventually a decision must be made to buy one type of paint and to complete a wall – and if we change our minds after that, it will be more expensive.

We must identify where we need flexibility and innovation and where changes are likely to be unimportant for overall value. Some markets, products and elements within a product change faster than others. Preston Smith points out, for example, that the electronics in an aeroplane change much faster than its structure.

It is not easy to decide where flexibility is essential and where it contributes little value – especially on subjective matters. When are last minute changes to a web design ill-considered interference? When do last minute changes represent an essential attention to detail that characterise respect for one's own brand? Everyone will have a different opinion – often depending on whether you are the manager being asked to approve this, the project manager, or the designer…

However, discussing such matters up front is important because it allows you to assign a cost to change. It doesn't mean that you can't call a halt to the project the day before launch and insist on a complete redesign of the site, merely that you should know the cost in both time and money when you do so. It also means that there is a duty to involve all stakeholders sufficiently early on. There is no point in showing your CEO the web design the day before launch if the only feedback you are prepared to hear is: marvellous, great job! If your CEO is likely to have strong views on user experience and architecture then she must be shown the web design at a far earlier stage.

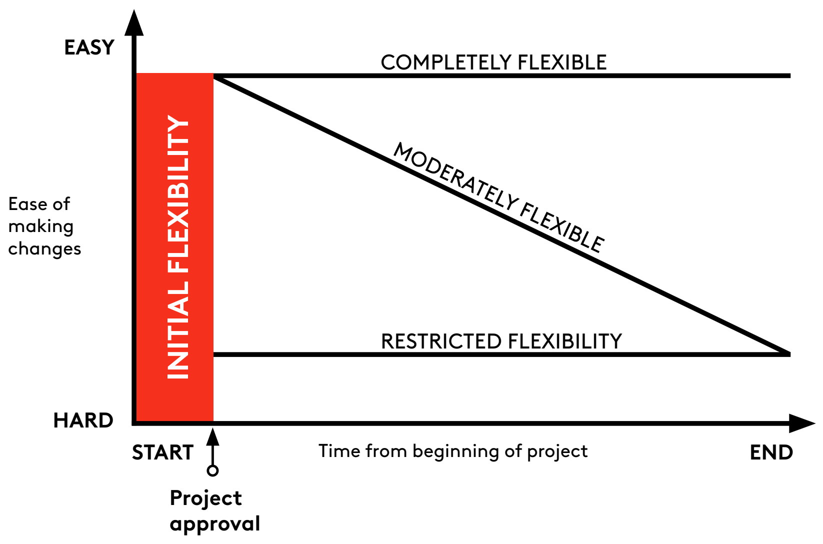


Figure 13. Three levels of managing flexibility in a development project

Restricted flexibility is the model most familiar since it fits with Waterfall development processes. Preston Smith describes it as, 'At the outset, the project has complete flexibility, since nothing is fixed yet. But at the end of initial (planning) phase, the project budget, schedule, and product requirements are established and approved. From here on, this project has restricted flexibility.'

Complete flexibility is the other extreme, where everything remains possible at all times – it is unlikely to be useful since decisions do need to be taken if the project's cycle time and budget are not to expand infinitely.

The important area is the middle. Here decisions are made at the last responsible moment, or just in time as it is also known. Peter Jackson, director of the multi award-winning Lord of the Rings trilogy, spoke of how his continual editing and re-editing of the films caused consternation at the studio. He chuckled over his producer's anxiety that he might deliver the film late. 'Of course we can't deliver late,' he emphasised, 'that would be irresponsible – but delivering just in time – that's the trick.'

Preston Smith expands on how to manage 'the trick': "decisions are made when necessary, often by progressively tightening up tolerances on variables. Thus the ability to make changes narrow methodically as development proceeds."

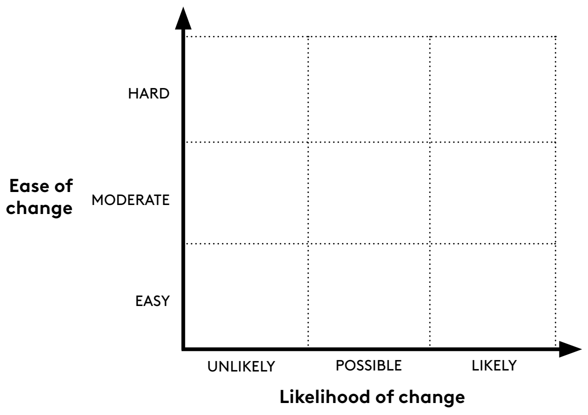
Activity 4: Identify where you really need flexibility

This activity should take about 15 minutes.

Wouldn't it be nice to be able to change any aspect of our projects at any point in time without incurring additional cost? In reality, as we progress and more software is added to the solution, certain areas of the solution begin to become calcified thus making them more difficult to change at a later date. We attempt to maintain flexibility in areas important to do so, but this has an associated cost. It is therefore important, like in the Peter Jackson story that you might have just read, to identify where flexibility is most important and where potential changes would mean significant investment.

Pick a single project that is currently being developed in your organisation. Identify different activities on the project that may or may not need change in the future. Select 10 of those and list them on a piece of paper. Try to be specific, you can include items like: "User interface for the card selection screen", "Content approval workflow", "Database tables for the user registration", "Charges calculation algorithm", …

For each of the items you have identified find out or estimate how likely the item might need changing in the future (unlikely, possible, likely) and how easy it is to change (easy, moderate, hard).



Plot all the ten items in the diagram.

Ideally, we would like to see things being likely to change also being easy to change, and the things hard to change being unlikely to change. Is that the case in your diagram?

If you have identified aspects of a project which are currently hard to change yet also likely to change in the future, pick one of them and make the necessary changes to increase your flexibility in that area.

In software development the three key areas that require flexibility are most likely to be: Requirements, Architecture and People.

Requirements

This is an area about which you are undoubtedly aware because it is perhaps the most famous element of agile development processes. While waterfall begins by creating an exact scope and then a plan that will deliver the scope, agile flips this on its head. Agile sets time-boxed development iterations and makes the requirements in each batch fit the time. This means that scope becomes flexible, with only what can be committed to immediate development entering the box.

The rationale behind this methodology is that software requirements need to remain flexible because customers' needs develop and evolve over time. A customer may not know exactly what they want at the beginning. The outlines of their true needs will emerge as the project takes shape. Rather than trying to force this early in development, locking down requirements and then measuring success against a theoretical need rather than an emerging actual need, flexibility demands that requirements can be formed and reformed throughout the process. In Lean Startup Eric Ries describes 'pivoting' as he and his team discovered that they had made incorrect assumptions about customer needs and had to change direction entirely.

Architecture

Over the years conventional practice in systems investment has prioritised delivered functionality and speed of delivery and ignored full life cycle costs and the ability to respond to change. IT infrastructure is typically under utilised, slow and expensive to change with very high levels of human resource involvement.

Our Integration session explores this issue in more detail, but here it should be highlighted that architecture is one of the most important areas in IT which needs to address the need for change. Most businesses now rely upon IT systems in order to function. Legacy systems can be unwieldy, requiring so much time to maintain and fix, that no resource is left to make changes or innovate. One way to deal with this is to try and build architectures designed to assist flexibility through being loosely coupled, allowing elements to be 'plugged' in and out. This may be the ideal – it is rarely so straight-forward in practice, since a system's complexity has emerged for very good reasons.

Trying to build flexibility into a system by adapting legacy systems often ends up in duplicating functionality and large management costs. Deciding whether to do this or to attempt to decouple the system as a whole to take account of future change and requirements is not always easy.

SOA is the solution most frequently discussed to attempt to balance future against current requirements, because it can be implemented in an incremental fashion. Like all other small batches, it keeps risk low (in comparison to replacing an entire system in one piece), allowing a company to reassess and reprioritise along the way.

When not to be flexible

Such emphasis has been put on flexibility by proponents of agile methodologies, that it risks seeming a panacea. We hope we have emphasised that there are other choices that can be made within flow. But besides those choices, there are also pitfalls to flexibility.

- **chaos** – too much flexibility, or flexibility too late in the process can feel extremely chaotic. Have you ever had a customer or boss who changes his or her mind every week? Moving from uncertainty to gathering certainty is the kind of progression we hope for, accompanied as it may be by a sudden need to rethink strategy based on feedback. Vacillating between opposing choices is not progression – it simply means that the project can get stuck with a lack of clarity in how to proceed. At this point the project needs to generate real, external feedback in order to point out a path.
- **unnecessary change** – change is designed to add value to the customer and flexibility allows us to make these valuable changes. A manager who says 'I really don't like the colour red – change the background' is not making a change based on customer value.
- **half-done work** – a belief that everything may be about to change can be viewed by some as an excuse to do sloppy work believing that someone else will fix it later.
- **change addiction** – we highlighted that people often like to work in a very flexible way. This can lead to skipping the due diligence required in planning and scheduling, and emphasise tactical reaction over long-term strategy. Typically a team that gets into this way of working will pour scorn on 'boring' plans and positively delight in lurching from one crisis to the next. Once again, this is not flexibility designed to provide value for the customer and is counter-productive for both the project and the organisation.

So, having identified where we need to be flexible, how flexible we want to be and having checked that we have chosen correctly, we turn to examining how we might use our tools to maximise our flexibility.

Batches, WIP and concurrent processing

Flexibility relies on small batches. A key function of small batch size is the ability to set up a fast feedback loop and fast feedback is the whole purpose of being flexible, since it enables us to act on the feedback to make a change. If we showcase the first step in a new feature to a customer only to discover they are not sure whether they will use it or not, we will want to switch onto a different feature rather than continuing to invest in something which is clearly not essential.

XP and Scrum control batch size through time-boxing each iteration. XP begins with a one-week batch size, which may extend in size from there. Scrum limits batch size to 30 days, although in practise for many companies this is reduced to 14 days.

Lean Startup focuses on the minimum viable product which in itself is an encouragement to reduce batch size by cutting unnecessary scope. Similarly Kanban has no explicit batch limit but uses WIP limits which tend to act as a means of reducing batch size.

> "By performing work in small, complete steps, JIT in the software arena gives us the ability to change direction at the end of each small completion—without any wasted effort."
>
> **Shalloway, A., Beaver, G., Trott,** J., 2009. Lean-Agile Software Development: Achieving Enterprise Agility.

The small steps mean that we can capture the value and move on to something else – using feedback to decide our direction. To do this we need to use small batches, an explicit limit to work in progress and usually a pull system to ensure that upstream processes only begin work when a signal is received from a downstream process. For example – a user story will be analysed just before it is built and validated – very different to a requirement with a full specification which is pulled into development months later and then waits for more months before it is tested along with all the other fully built features.

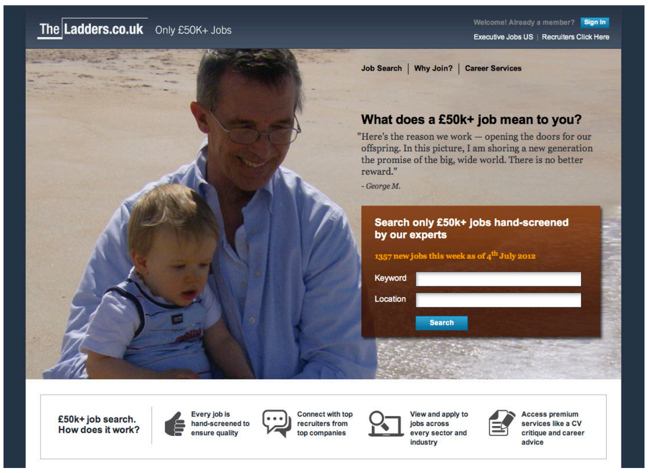
The small batches and WIP constraints mean that there should be very few queues within the development process. But there will be a large queue just before work is pulled into it – known in agile as the project backlog. M. Kennaley, in his book SDLC 3.0: Beyond A Tacit Understanding Of Agile, describes the backlog as "a customer managed queue of demand requests for the product."

Because sequential development almost always leads to large batch sizes, it is not suitable for flexibility. As early as 1986, the Harvard Business Review commented on one of the most famous development processes of the time, "The traditional sequential or 'relay race' approach to product development—exemplified by the National Aeronautics and Space Administration's phased program planning (PPP) system—may conflict with the goals of maximum speed and flexibility."

The concurrency of overlapping analysis, design, development and testing is essential for the increased speed and small steps that permit the fast feedback loop.

CASE STUDY: The Ladders recruitment site

The Ladders is a job search recruitment site which operates a subscription model helping candidates find higher-paying jobs (£50k +).



What does a £50k+ job mean to you?

"Here's the reason we work — opening the doors for our offspring. In this picture, I am shoring a new generation the promise of the big, wide world. There is no better reward."

- *George M.*

Search only £50k+ jobs hand-screened by our experts

1357 new jobs this week as of 4th July 2012

£50k+ job search. How does it work?

Every job is hand-screened to ensure quality

Connect with top recruiters from top companies

View and apply to jobs across every sector and industry

Access premium services like a CV critique and career advice

Figure 14. A screenshot from The Ladders website

A site which requires constant interaction, whether employers or recruiters posting jobs or candidates searching and applying for them, means that the user experience is essential at every step. Since user interaction is so key, that in itself militates a short feedback cycle. The teams need to know whether each new feature or piece of functionality really is helping a customer or user. If it isn't, then the team needs to swiftly change direction or adapt their work. Such needs led the company to decide that the priority in development should be flexibility.

Previously, the team had used a waterfall method with great importance given to the UX designer who created a detailed set of deliverables, including workflow diagrams, sitemaps, wireframes, annotations and detailed UI specification docs. Since the designer was working alone, when the ideas were presented back to the wider team there would often be conflicts – for example, what developers thought they could build or integrate with the rest of the system or what managers thought was cost-effective. This was not only time-consuming, but it meant everyone was dissatisfied – the original 'purity' of the design almost never actually got deployed, while those building it felt little ownership over the designs.

Eventually the company decided to use the Scrum methodology as a way to increase flexibility. Rather than a single designer presenting solutions to the build team, the entire team was involved in the design process from the start (a product manager, 4 developers, a QA analyst, UX designer and a dev manager).

Ensuring there was less time and effort put into the initial deliverables was crucial. Rather than a designer coming back with a fully worked out solution including a beautiful mock-up, the team favoured sketches, hacked screenshots and rough prototypes. Similarly, rather than documenting all decisions and flows, most details would be agreed in team conversations based around demonstrations rather than formal reviews. As feedback was gathered more quickly, a new design version could be delivered in hours rather than days.

Soon the team was deploying updates every two weeks. This meant they could try things out, knowing that if they failed, the next iteration to correct or adapt the decision was only two weeks away. This in turn allowed senior managers to increase their trust in the team and thus the team's autonomy. The goal for the team was to 'solve communication problems between job seekers and recruiters' and the team could select KPIs against which they could measure the success of each change. For example, did the response rates and number of communications sent through the system increase? Based on the answers to such questions, the team could apply the learning to adapt the solution in the next iteration.

Note that while launches got faster and more effective, which undoubtedly brought financial benefits to the company, at first there was a period which must have felt uncomfortable as the whole team was committed to working on problems which previously would have been the preserve of just one or two people. This must have initially felt very resource hungry, perhaps even wasteful. Only by committing the resources up front were the team able to reap the eventual benefits.

Summary

The entire aim of flexibility is to respond to valuable customer needs in a timely manner. When this is judged correctly, the results are excellent. Of course if the customer need turns out not to be valuable (producing fridges in 20 different colours when customers are happy with white), the increased costs of flexibility will prove painful. There are a few side benefits as well, however. Increased flexibility often produces fewer defects because of the fast feedback loop – and this can often offset a great deal of the cost associated with creating increased flexibility.

4.3. All out speed

On the far left of the graph, companies are prepared to pay high costs in order to achieve very fast cycle times. Let's begin by emphasising that this is not a common flow choice in IT and that we shall therefore cover it more briefly.

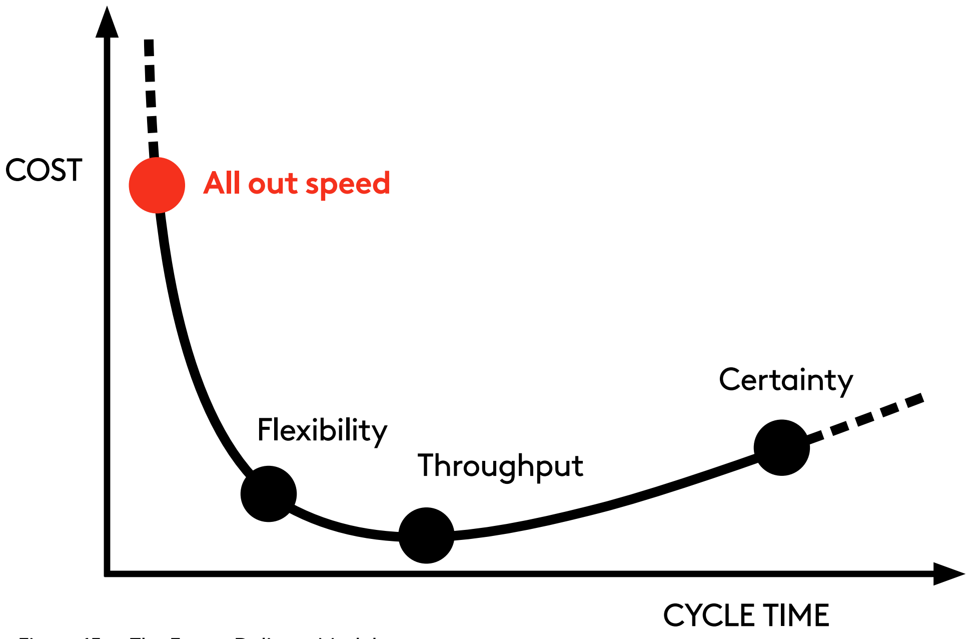


Figure 15. The Faster Delivery Model

Think of an ambulance racing through the streets with blue lights flashing and siren wailing. It does not have to obey usual traffic rules, it is able to bypass queues and jams, while other traffic moves out of its way. The paramedics in the back are engaged in potentially life-saving procedures, but they are also communicating with the hospital ahead – warning them to prep the operating room; giving the patient's blood type so appropriate transfusions can be ready. They are making detailed notes to hand over to the hospital team to ensure there is no delay or missed information in appropriate treatment.

An ambulance focuses on one thing only – getting its patient to hospital. It ignores anything off the critical path – that means an ambulance is not sent out unless it has been serviced, filled up with fuel, its inventory checked and its complete team are ready and able to work. All such work must happen before the ambulance goes 'on call'. That means even if the emergency services had a quiet night, the driver wouldn't fill in time checking the oil. In order to be ready for an emergency, a certain level of slack has to be built into capacity. Other faster processes we described are very much in evidence: part-time capacity is reflected through shift work as well as being able to 'switch'

Figure 16. Ambulance on an emergency

over teams from adjacent areas. During the London riots in the summer of 2011, not only police were called in from nearby counties, but also ambulance teams. Sequencing of the queue is essential to the emergency services and depends on strictly predefined rules to guide telephone staff as to which priority to assign to each call: a heart attack will get a faster response than a woman in labour; a call from a medical professional to 'bluelight' a patient to hospital will get a faster response still.

The high costs of speed

The example showcases many of the defining features of all out speed. It tends to be used rarely – and no wonder. Costs are high, productivity is very low and the risks are also high. We reserve it for emergencies or because the customer demand is very valuable to us and/or there is a high cost of delay. It is sometimes referred to as an 'expedite request'. A business case justification should always be a requirement to explain how and why the costs of all out speed are required.

While occasionally all out speed is justified by an opportunity – needing to leapfrog a competitor's launch, for example – more often it is in response to a threat or an emergency that may already be costing the company money. Severe production defects might not only jeopardise sales of a product but they might cause a crisis for the company's reputation that requires an immediate, costly fix.

Because the value or the threat is so high, time becomes the only crucial factor to which all other considerations are secondary. While having lots of extra capacity might seem like one likely expense, it is not the only one:

> "To overcome project uncertainties and unexpected problems, you must develop alternative solutions in parallel, and the fastest solution usually wins."
>
> Shenhar, A. J., Dvir, D., 2007. Reinventing Project Management: The Diamond Approach to Successful Growth & Innovation.

Developing parallel solutions or set-based concurrent engineering, as it is also known, can be extremely expensive. During the development of the Apple Mac Steve Jobs demanded the team develop a drive with a small company called Alps. His team were convinced that the drive would never be ready in time. They disobeyed his direct instructions and developed a drive with Sony in parallel in secret. Once this even involved shoving the Sony engineer into the cupboard when Jobs visited the office. When – as the team predicted – Alps confessed that it would take them a further 18 months to get the drive into production, the team announced they had an alternative solution ready to go. Jobs is supposed to have sworn when he realised what had happened – not in anger, but in amazement at his team's daring and resourcefulness. Just imagine the opposite possibility for a moment – Alps did produce the drive in time, and Steve Jobs found out his team had disobeyed him and had spent a large amount of development money in the process… quite a few people might have been seeking alternative employment.

There is yet a further cost to be factored in to all out speed – its impact on other projects. Since these are bypassed by the expedited project, they will be delayed and suffer the attendant cumulative cost of delay. All the cars that the ambulance passes have to slow down and get out of the way, for example. While each may suffer only a small delay individually, the overall cost of delay to all of them is large. David Anderson writes regarding this cost in his book Kanban, "It affects predictability of other requests. It increases mean lead time and the spread of variability and it reduces throughput… Expediting is undesirable even if it is being done to generate value."

Using the tools that expedite all out speed

Queues

Since queues cause delay, all out speed requires no queues. Just like the ambulance, the project that uses this flow choice must have a great deal of background work done to support it.

Work often goes on around the clock with engineers working in shifts. Ensuring seamless handovers may mean that engineers even work side-by-side while completing a handover (effectively doubling your staff costs). Since constant interaction is necessary, senior management also need to hold themselves in readiness to be present at all discussions where a key decision must be made.

Batches

During the Apollo 13 crisis where the shuttle's fuel tank exploded, there were only a few hours left before oxygen and power would run out. The commanders had to separate the engineers to work on one task each. The same person could not consider how to reboot the shuttle's system and how to build a filter to remove carbon dioxide.

Since small batches facilitate faster processes, all out speed normally requires batch size to be small – but not always. The Apollo engineers were working on a clearly defined problem – but each problem still encompassed a series of complicated steps. In the case of the carbon dioxide filter the solution had to be a single batch – adapting the lithium batteries from the command module to fit the lunar module could not be broken down.

In all out speed, because all systems must be set up to allow a batch to be walked through as one-piece flow, a larger batch can actually be tolerated without creating a large delay. The Apollo 13 engineers designed a filter connection using the crew's available materials, tested it, gained immediate approval and then communicated the solution to the astronauts.

Figure 17. Apollo 13 engineers

However, a small batch remains a good way to reduce the high risk of working at such speed – any change can be tracked and mapped, a wrong direction can be corrected more swiftly. Software developers should be grateful that an error in design is unlikely to be shown up by your crew going unconscious and dying.

WIP

The ambulance has one patient. If there is another emergency ahead, maybe a car crash a street away, the ambulance does not divert to pick up a second patient. Transporting two patients might seem sensible to those considering efficiency, but the ambulance has no such concern. A WIP constraint of one is the ideal work limit for all out speed. A second item would cause a potential delay which defeats the point of all out speed.

Companies have to be rigorous about enforcing such a measure if they are serious about the importance of the project. Only one 'expedite' request can be allowed at a time, for example, while all the other work in the system goes on hold. Thus pre-existing WIP is carried as a queue.

Capacity

When David Anderson writes of expedite requests he states that 'Capacity is not being held in reserve for expediting.' Instead he believes simply leapfrogging the queue will be enough. For a crisis or even a project planned to deliver at all out speed, this may not be sufficient. People may be pulled off other projects, the original team may stay late night after night, perhaps part-time staff are called in or trusted consultants hired. If the project is a planned expedited request, spare capacity should have been factored in as an acceptable cost from the start.

Concurrent and faster development processes

In the ambulance the need to avoid delay through concurrent processing is why the paramedics radio ahead to inform the hospital when they will arrive and ask them to prep the operating room including fetching blood of the correct blood type.

Crucial to all out speed, concurrent processing is initiated wherever possible. Indeed, because all out speed requires one-piece flow the communication and interaction (usually the hardest element of concurrent development) is often much easier. Nonetheless, companies have to ensure the systems are in place to ensure all appropriate departments understand the 'expedite request' and that it is not abused. There is little point in having an expedite class if individual managers are good at bribing, bullying and begging their way to the top of the queue with every request that has personal rather than business-wide significance. It's a style frequently shown in the popular TV drama CSI, for example, where especially attractive officers often manage to have their murder investigation samples tested in advance of other cases…

CASE STUDY: Brawn GP in Formula 1

Formula 1 design teams need to work almost as speedily as their cars perform, while their product life cycle (design-build-test) has to be extremely short in order to see improvements that can be implemented in cars raced in specific competitions.

Figure 18. Brawn GP racing car

Brawn GP was a relatively small motor racing team based in the UK who participated in the 2009 FIA Formula 1 World Championship. Competing against teams with deeper pockets, they relied on different elements to give the team an edge. A major change to regulations provided an opportunity. At this point all the teams need to make large-scale design changes rather than just tweaking last year's model. A team like Brawn, who had great creativity and fast development processes, hoped to be able to make an improvement that would edge ahead of the competition. Naturally the team needed to design and build quickly in order to hit the next race. The earlier designs were tested, the more time there would be to allow tweaks and performance enhancements to the car. In such a case all out speed is clearly the most important development choice.

In 2008/9 a sweeping regulatory change provided an opportunity Brawn GP was determined to seize – right when the team changed to new software design systems. Because timelines were so tight, both old and new systems were run in parallel while the transition occurred, including training designers with the new tools at the same time as they were actually designing and building the new car. Naturally the team used concurrent design and engineering processes for the design and build itself. Sometimes this unavoidably led to re-work (or last minute changes required), for example, when the team had to install a Mercedes-Benz engine just seven weeks before the start of the season. The software tools they used were designed to help them stay flexible for such issues – relational design helped them make the required changes to related systems without needing to worry about unforeseen knock-on effects.

At other times all out speed led to additional costs. The manufacturing team needed to produce 14 gearboxes. It knew it didn't have the capacity and so outsourced seven to a more expensive external manufacturer. Even the resource-heavy external manufacturer struggled with the last-minute changes to design, however – something that probably pushed cost up even further.

All this was necessary because a faster development cycle meant that the team was able to test the car more times, whether in the wind tunnel, with computational fluid dynamics, or on the track – and then make the necessary adjustments and tweaks and test again. The result was an extremely successful run in the 2009 season with unprecedented success against much better established rivals.

Summary

Most of us intuitively understand all out speed – we are familiar with it from films and our own lives. A crisis! Drop everything! Work on this! Just get it done! The feeling can be exciting, people are prepared to pull out all the stops and there is an enormous sense of achievement after managing to pull off the seemingly impossible timescale. But there is a danger involved in it – such excitement can be addictive and can lead to a company that lurches from crisis to crisis. All out speed is extremely expensive, it is rarely sustainable and it is risky – it often leads to mistakes as tired or pressured teams push out a project that is improperly checked. If used as an excuse for sloppy work, all out speed can be not only expensive, but disastrous too!

4.4. Certainty

We gave you a fairly robust critique of companies who like to insist on 'certain' or 'on time' delivery above. That doesn't mean that it is NEVER the right decision, but it is very RARELY the right choice.

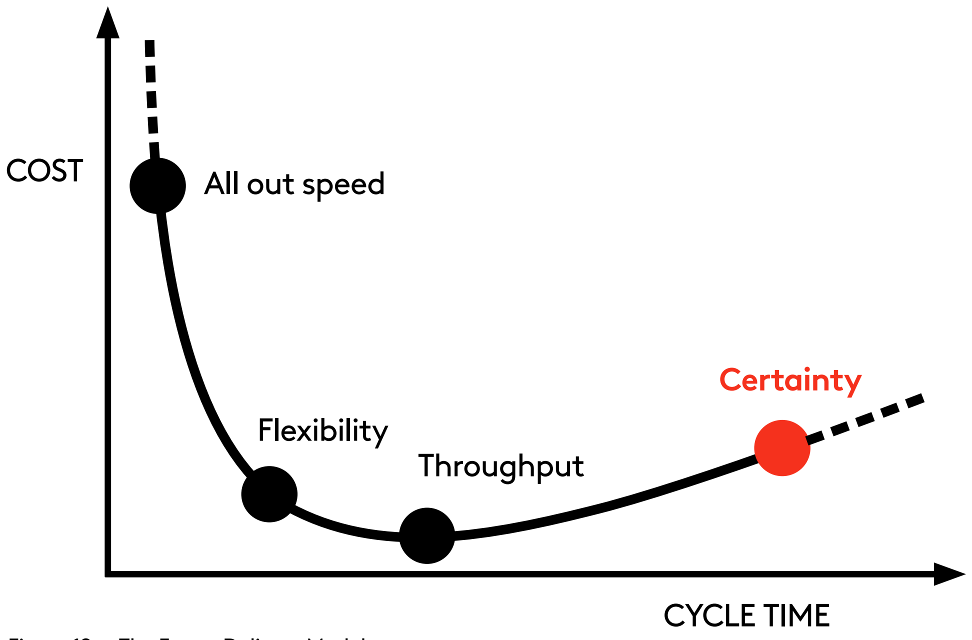


Figure 19. The Faster Delivery Model

When is it right?

For certainty to be the right choice we need to have certainty of input to match the certain output to which we are committing. Since this is rarely the case in true software development (by its nature creative and iterative), it means very unambitious requirements – things which we absolutely know we can already do and can provide highly accurate assessments of resource they will require.

Normally there must be a real customer value attached to a specific window of delivery. It also means that there is little benefit in delivering early, that valuable dependencies are high, and that the work cannot be delivered in part.

When athletes arrived at the Commonwealth Games in Delhi in 2010, there was an enormous outcry that the construction had not been finished properly in time. There was no alternative plan – the athletes had to stay somewhere. Nor was anyone impressed that most of the accommodation was finished. The company did not get cut any slack for the fact that faulty apartments were nearly finished.

Figure 20. Stadium for the 2010 Commonwealth Games in Delhi

This is a good example of a genuine occasion where certainty was probably the right choice.

You will notice that the example is taken from construction – not IT. Imagine we're creating the timing system for the Commonwealth Games – there's still an enormous number of dependencies, the press will still crucify us if we fail… and yet certainty is NOT the right option.

Because the problem can be split up and value can be obtained from early delivery, we could batch up the issue by sport (or stadium). We could perfect the automatic timing system for athletics, before moving onto cycling. We could ensure basic functionality – recording of the times, and then move on to other features – automatically sending the times to the media and display boards. Part of the solution will still provide value. That means that focusing on throughput is the right choice, still planning backwards from our deadline, and still with contingency plans in place (people who will manually enter the times onto the display board), but without creating the slowest and least ambitious programme because of a mistaken focus on certainty.

In order for certainty to succeed as a flow choice, senior managers must understand the consequences of the choice. Any desire to make changes further on in the process will imperil certainty. If management want an ambitious or innovative project then certainty is not the correct choice.

Most companies who fixate on certainty are making a poorly-considered default choice. The overwhelming temptation when looking at your organisation is to say 'ah yes, but my problem is very complicated, it does have a valuable customer demand, and there are huge dependencies'. Of course this is true – but just as with our imaginary timing system for the Commonwealth Games – you can normally do something about these. Once you are really convinced that certainty is wrong, you have already cleared the biggest hurdle. Breaking down the problem, discovering smaller batches of value to deliver – this is actually not impossible once you begin to work on the problem.

The tools that assist certainty

Using our tools effectively will still help optimise flow.

There will always be uncertainty in any project. We deal with uncertainty by creating buffers. These buffers take the form of capacity, money and time. The most important element of managing for this flow choice is to guard those buffers jealously. They are there for emergencies and unexpected difficulties and not to permit scope creep or even flexibility.

Queues

Queues mean extra delays. To this end measuring and controlling queues will help achieve more control over the process – essential for those with truly immovable deadlines. Any emerging queue must receive immediate attention to stop it spiralling out of control.

Batches

Since certainty requires detailed planning, the emphasis is often on writing very detailed requirements and exact specifications. This leads to large batches at the beginning (all the specification must be done before approval when any design can begin) which then continues throughout the process, often culminating in 'big bang' integration. Problems at this point can involve enormous quantities of rework – just when, according to the schedule, the project should be ready to go live. Thus large batches normally increase risk.

Reducing batch size should assist in revealing actual progress for the project thus increasing predictability. This is the case even when the project must all be delivered in one release. There is still value in delivering increments early, even if these increments are not released to the public.

However, one of the key benefits of batches – flexibility – does not hold true here. Flexibility means uncertainty and that will militate against security in scheduling. If there is any chance that flexibility is going to be of value to you – then you have made the wrong flow choice.

WIP

By making WIP visible it becomes easier to spot queues and then manage them. This is essential to avoid situations that could derail the project. Despite this, projects with detailed and immobile schedules are the ones least likely to use task boards to be owned by the team – they are far more likely to rely on spreadsheets and plans on the wall in a project management office.

Figure 21. Underground train held at station

WIP limits minimise schedule variation to reduce the likelihood of queues building up at all. This is adapted from congestion theory – and can be seen in action every time two buses arrive at a stop together. In order to see how WIP limits can improve flow, all you have to do is travel on the London Underground a few times. It won't take many journeys before you hear a voice crackling over the tannoy: "This train is being held at the station to regulate gaps in the system". If a train stops at a crowded station where lots of passengers have to get on an off, it will inevitably travel slightly slower. As it gets slower there will be more time for passengers to accumulate on the platform of the next station causing it to have to stop for longer here as well. Meanwhile the train behind it with fewer passengers to let on and off will be moving slightly quicker. Eventually the two trains will bunch up together. To avoid this, an early train will be held at the station, collecting a few extra passengers, for a set time and will then be released. It makes complete sense for the overall flow of passengers in the system as a whole (but that doesn't stop it being annoying when you're on the train in question).

We can do the same thing for WIP – by reducing the amount of WIP in the system and releasing it only at regular intervals, we regulate flow. Luckily unlike passengers, tasks do not complain about being held at a station.

Capacity

Spare capacity needs to be switched in order to tackle emergent queues. Note that this spare capacity should be available throughout the development process. Queues are just as likely to exist near the beginning of the work, while having people who have worked on and off for the project over a lengthy period of time are more likely to able to switch in usefully and become swiftly productive at crisis points. Unfortunately the common approach is to add a vast crew of people at the very end of the development process only when it is clear the project as a whole is in trouble – thus triggering Brooks's Law that adding more manpower to a late project makes it later.

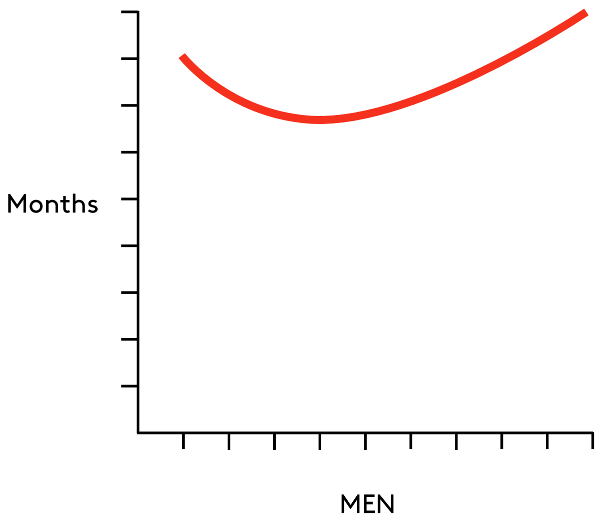


Figure 22. Brooks's Law from The Mythical Man-Month, 1975

Faster development processes and concurrent development

Concurrent development carries with it an inherent information risk. If we begin design before we have finished our analysis, we may develop features we don't require, or spend ages on a feature we later decide is of minimal significance. This works all the way through the value chain – imagine a marketing manager briefing an advertising campaign around a key 'delight' feature, only to discover that in final testing engineers have decided to jettison this element because it isn't working.

The only way around this would be strict sequential development with a 100% batch at each stage. Many organisations claim to work like this, but in fact the timescales required for it are unacceptable. This means that unofficially there is a great deal of overlapping processing, but because it is unofficial the risks of elements clashing with one another are not well managed. The more processes overlap, the more two-way feedback is required – a principle that sounds obvious, but which is so little applied that the industry is littered with examples of a lack of information transfer.

"Because the information is incomplete, communication must go both ways as recipients ask questions about the data to find out what it means and provide feedback as to how well it meets their current needs. Using partial information has its price, because we occasionally make incorrect assumptions when applying it and then some work has to be redone."

Smith, P., Reinertsen, D., Developing Products In Half The Time: New Rules, New Tools.

Summary

Slow, inflexible and rarely the cheapest option, a focus on certainty is not a good choice. Giving up certainty does not mean giving up predictability, while most projects with external deadlines will still benefit from delivering working increments early. If despite all such warnings you are convinced that this is the right choice for you, considering the tools that can optimise end to end flow is more important than ever. No-one wants to have worked on an IT project held up for public outcry for being embarrassingly late and over budget.

5 CONCLUSION

Faster delivery is something that most companies desire. Managers know the benefits of being early to market, of responsiveness to customers, early revenue and faster payback on investment. In order to achieve these tempting benefits they are prepared to try out different methodologies and invest significant amounts of money in training and tools. Worryingly, their efforts often appear to make little appreciable difference, so the search continues for the next 'how to' manual that promises to solve problems and increase the speed of delivery. Because these managers are already frantically busy, they yell out 'just tell me what to do!' The world being what it is, a host of training providers, coaches, authors and associations spring from the ground to provide the answer.

This doesn't work. The problem isn't the tools and the methodologies – they are filled with good ideas based on sound principles. The problem isn't that those offering solutions aren't bright, honest people with integrity. Nor is the problem the managers or their teams being stubborn or stupid in some way… It's rather more fundamental than this – the way we act is based less on rational logic than on deep inner instinctive ways of thinking, and we don't change that based on a how to manual.

In the majority of cases, those who have taken large steps forward in their speed of delivery are using a hybrid of differing methodologies that they have developed through trial and error. They have examined the difficult trade-offs to be made, and tested possible answers until they come up with what works for their unique situation. Will I sacrifice flexibility for efficiency? To what lengths will I go to incorporate real customer feedback? How am I going to break this project down into units of value? Can I decouple the processes? What will persuade marketing to agree to my new uncertain schedule and just in time decision-making?

In order to institute change (and change is hard) then you need to understand why you are doing it and why it works. You need to be so convinced of it that you will push it through in the face of grumbling and resistance – not just from colleagues, but from your own gut instinct.

We have seen companies practice a rigid adherence to the letter of an agile practice while completely missing the point of the exercise. Writing your user story on an index card, for example, is intended to keep things so deliberately brief that the developer is forced to have a conversation with the product owner and tester and flesh out the idea mentally. It's intended to encourage responsibility, creativity and collaboration. We've known cases where people write a traditional product specification in very small writing on an index card and then congratulate themselves on how agile they are.

User stories, stand ups, scrum masters, coaching, task boards … none of these things matter in and of themselves – they are tools to help you visualise work, improve collaboration, or remove obstacles to flow. If they are implemented without their underlying purpose being understood or appreciated then they are unlikely to bring about significant change or improvement. A commitment to improving flow and delivering early may sound more nebulous, but in fact it is a far more solid foundation on which to build.

Five things to do this week:

1. Measure cycle time. How long from when a request enters the system to when it goes live? Track that request. Where is it spending the most time – is it in a queue?
2. Visualise work. This does not have to be a fancy task board. Simply ensure everyone's to do list is written down as a single action per post it note. Put these on the wall. What does it tell you about your flow?
3. Try a WIP limit. Persuade the team to test a WIP limit for this week only – or even for a single day! What happens when someone who has finished 'their' task needs to help another team member finish theirs?
4. Break down a batch. Pick a task and try to break it down. Track the cycle time of each element.
5. Select a single additional tool from the practice library and run it for a week.

Learning outcomes

By the end of this session you should:

Appreciate the benefits of decreasing cycle time

- Release value early, improving revenue and payback
- Shorten the feedback loop, increasing responsiveness to customers
- Gain market share and awareness
- Increase productivity and make a reduction in costs

Comprehend a model of flow choices

- Key trade-offs between cycle time and cost
- The unprofitable extremes

Where your company sits within this model

- The reasons we choose 'certainty'
- The effects of certainty on cost and time through buffers
- The illusion of certainty

Appreciate alternative Flow choices, their benefits, most likely application and the tools which best promote them

- Throughput – maximising efficiency where there is up-front knowledge
    - Buffers and queue control
    - Standardisation and early testing
- Flexibility – dealing with uncertainty
    - Change costs and this cost rises over time
    - Identify where you need flexibility
    - Focus on small batching, fast feedback loops
- All out speed – most common for emergency problems or threats such as correcting public mistakes
    - High cost, rarely sustainable and high risk
    - Requires one-piece flow, zero queues, concurrent processing and excess capacity
- Certainty – certainty of input and a real customer demand for a specific window of delivery
    - Buffers of time and cost provide certainty making the project slower and more expensive
    - Small batches and lower WIP provide more control and increase predictability
    - Certainty is rarely the right choice in software

BIBLIOGRAPHY

Ambler, S., Examining the Agile Cost of Change Curve. [online] Available at: <http://www.agilemodeling.com/essays/costOfChange.htm>. [Accessed 9th February 2012].

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

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

Boehm, B., 1981. Software Engineering Economics. Prentice Hall.

CBDI Report, 2005. Business Flexibility Through SOA. [online] Available at: <ftp://ftp.software.ibm.com/software/soa/pdf/CBDIWhitepaperBusinessFlexibilityThroughSOA.pdf>. [Accessed 9 February 2012].

Cohn, M., 2005. Agile Estimating and Planning. Prentice Hall.

Denne, M., Cleland-Huang, J., 2003. Software By Numbers: Low-Risk High-Return Development. Prentice Hall.

Dov Dvir., Reinventing Project Management: The Diamond Approach to Successful Growth & Innovation.

Goldratt, E., Cox, J., 200**4. The Goal: A Process Of Ongoing Improvement. 3rd Revised Edition. Gower Publishing Ltd.**

Harvard Business Review, 1986. New, New Product Development Game. [online] Available at: <http://hbr.org/1986/01/the-new-new-product-development-game/ar/1>. [Accessed 8 February 2012].

Harvard Business Review, 2007. Too Far Ahead Of The IT Curve? [online] Available at: <http://hbr.org/2007/07/too-far-ahead-of-the-it-curve/ar/1>. [Accessed 9 February 2012].

Kennaley, M., 2010. SDLC 3.0: Beyond A Tacit Understanding Of Agile. Fourth Medium Press.

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

Labour Matters, 2011. Ken condemns London Ambulance Service cuts. [online] Available at: <http://www.labourmatters.com/london-assembly-labour/ken-condemns-london-ambulance-service-cuts/>. [Accessed 15 March 2012].

Manifesto for Agile Software Development, 2001. Principles behind the agile manifesto. [online] Available at <http://agilemanifesto.org/principles.html>. [Accessed 8 February 2012].

McConnell, S., 1996. Rapid Development: Taming Wild Software Schedules. Microsoft Press.

Project Management Institute, 2009. A Guide To The Project Management Body Of Knowledge: PMBOK Guide. 4th Edition. Project Management Institute.

Reinertsen, D., 1998. Managing the Design Factory: A Product Developer's Toolkit. Simon & Schuster Ltd.

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

Rico, D., Sayani, H., Sone, S., 2010. The Business Value of Agile Methods. J. Ross Publishing.

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

Shenhar, A., Dvir, D., 2007. Reinventing Project Management: The Diamond Approach To Successful Growth And Innovation. Harvard Business School Press.

Smith, P., 2007. Flexible Product Development: Building Agility for Changing Markets. Jossey Bass.

Smith, P., Reinertsen, D., 1998. Developing Products In Half The Time: New Rules, New Tools. John Wiley & Sons.

Standish Group, 2005. The Chaos Report. [online] Available at: <http://www.projectsmart.co.uk/docs/chaos-report.pdf>. [Accessed 10 February 2012].

VFQ Value Flow Quality

valueflowquality.com

emergn.com

by emergn