Value
Flow
Quality

Optimising flow

by emergn

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

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

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

Emergn Ltd.
20 Harcourt Street
Dublin, D02 H364
Ireland

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

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

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

CONTENTS

INTRODUCTION

Figure 1. Updated iPhone with copy and paste feature

When Apple launched its first iPhone, it came with a version of the OS X (yes, it really wasn't iOS) operating software. We all know how phenomenally successful it was. Thousands of people queued outside the doors to the Apple store to buy one; off duty policemen were hired to guard the stores at night to stop thefts, and within 74 days of the release, Apple had sold a million phones.

And yet despite this incredible success, the iPhone lacked several features that other phones had as standard. It could not copy and paste (say take a phone number from a message into contacts). The original iPhone camera was pretty rubbish compared to those on other phones. The software could not support MMS (which pretty much every other smartphone did). It didn't even support SMS – less important in the American market, but a major disadvantage in Europe. There was no GPS (Nokia and Sony plus several others had it), and the iPhone couldn't support any applications. To many companies, such omissions would have seemed an insurmountable problem. 'We must be able to match what our competitors do!' they might have cried, 'or no one will buy our product!'

Steve Jobs's genius, and arrogance, was to believe in the superiority of his product without extra features. He wanted to launch immediately in order to capture market share in a rapidly changing market. Other features were launched later as they became available, as were fixes to various problems. Apple significantly ramped up the rate of innovation, launching new models and discounting the old as it went. When early adopters complained of feeling penalised, Apple didn't change their pricing structure, they simply offered those who complained store credit off their next purchase – an expression of true confidence not only in the strength of the brand but in customer loyalty.

Moving fast means more time in market and more awareness - and both of these together should add up to more sales. If you're first to market and first to capture the market share, you can raise the barriers to entry for competitors, leaving them a sad little bunch of 'me-toos' bobbing in your wake.

Of course, there are some risks associated with being first – since you're creating the market, maybe no-one will adopt your product; you have to spend the most on advertising, while a later launch will be able to ride on the awareness others have generated; you will also probably have bugs that competitors will be able to fix by the time they launch their product.

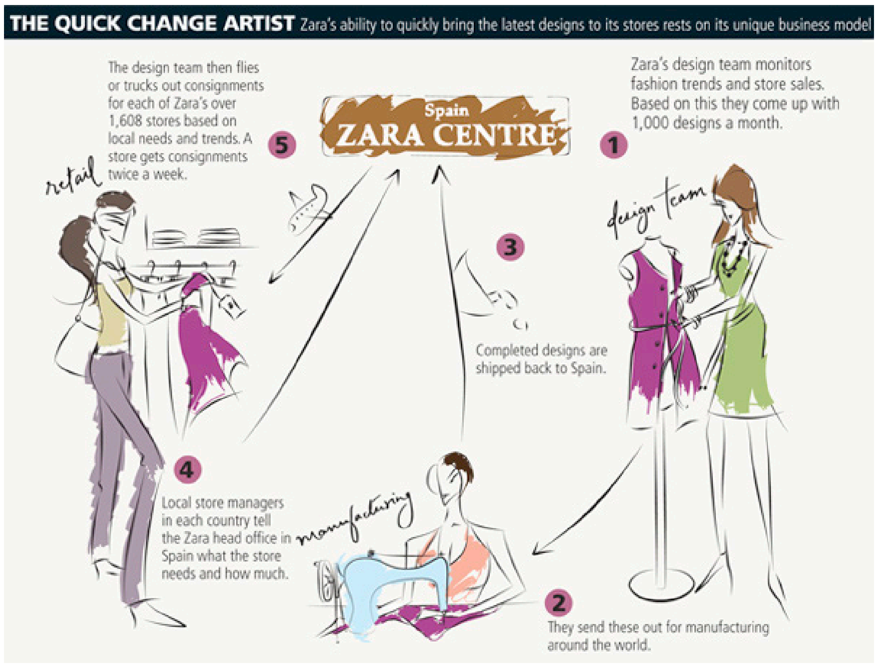

THE QUICK CHANGE ARTIST Zara's ability to quickly bring the latest designs to its stores rests on its unique business model
Spain
ZARA CENTRE
1 Zara's design team monitors fashion trends and store sales. Based on this they come up with 1,000 designs a month.
design team
2 They send these out for manufacturing around the world.
manufacturing
3 Completed designs are shipped back to Spain.
4 Local store managers in each country tell the Zara head office in Spain what the store needs and how much.
5 The design team then flies or trucks out consignments for each of Zara's over 1,608 stores based on local needs and trends. A store gets consignments twice a week.
retail

Figure 2. Zara's unique business model

Being first means moving quickly. Being second means moving even faster. A fashion designer may spend a year preparing their catwalk show. The high-street retailer that plans to copy the catwalk design (Zara is a famous example), has to recreate the design at an affordable price, ramp up production and deliver the clothes to stores within a few weeks. Similarly, moving fast (perhaps better described as being nimble or flexible) means you can respond effectively to changing conditions – a competitor's launch, a new technology, a change in standards or governance. In either case, moving fast has become one of the most important considerations for any company.

Delivering innovation at speed is so important that you will undoubtedly have sat in on numerous debates over plans, schedules and delivery dates, and perhaps read some of the multitude of books or methodologies designed to help speed up projects…

In this session we will examine what getting faster actually means in terms of the business as a whole – the methods, structures, attitudes and tools that support moving faster. Speed has a value to your business, but it also has a cost, so this session also briefly explores the idea of how you might balance these two.

Before we dive into tools to make us faster, we need to look at some truly basic questions about what speed might mean to us. Questions which, despite their simplicity, are rarely asked by those managing projects.

By the end of this session, students will be able to:

1. Appreciate why time is so important to an organisation and why the customer's view of time matters.
2. Consider the benefits and limitations of the 'on time' delivery approach adopted by most companies.
3. Take a broader view of end to end flow.
4. Analyse the cost of delay in development projects.
5. Appreciate the opportunity offered by flow optimisation to eliminate delay and increase delivery speed.
6. Judge how to optimise flow at scale.

1 TIME: THE UNIQUE RESOURCE

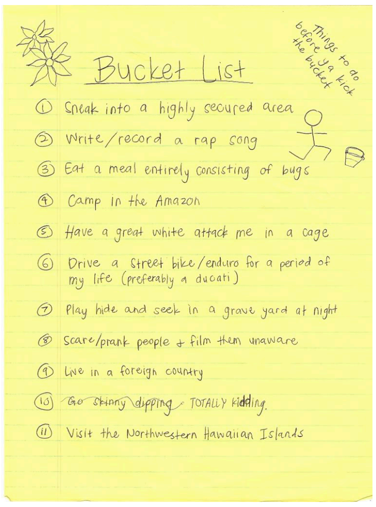
Figure 3. Peter Drucker on the cover of Business Week

"Time is a unique resource. Of the major resources, money is actually quite plentiful [...] People – the third limiting resource – one can hire, though one can rarely hire enough good people. But one cannot rent, hire, buy or otherwise obtain more time."

Peter Drucker

As Peter Drucker elegantly points out, time is inelastic – no matter how much I am prepared to pay for it, you cannot give me yesterday to spend over again.

Everything requires time. It is a universal condition. Even if we pay others to do something for us – even if they are expert in what they do – the work will still require time. And yet, if there is one thing that humans are astonishingly bad at managing, it is their time. Take the modern phenomenon of the so-called 'bucket list', a list of things to do 'before I kick the bucket' (meaning 'before I die'). It is only as time pressure increases with the awareness of death, whether due to age or illness, that we decide to first create, and then act on such a list. Just as our resource of time becomes scarce, we try to fit the most into it.



Bucket List

Things to do before ya kick the bucket

1. Sneak into a highly secured area
2. Write/record a rap song
3. Eat a meal entirely consisting of bugs
4. Camp in the Amazon
5. Have a great white attack me in a cage
6. Drive a street bike/enduro for a period of my life (preferably a ducati)
7. Play hide and seek in a grave yard at night
8. Scare/prank people + film them unaware
9. Live in a foreign country
10. ~~Go skinny dipping~~ TOTALLY kidding.
11. Visit the Northwestern Hawaiian Islands

Figure 4. An example bucket list

Activity 1: There's never enough time in the day

To Do lists, the favourite tool of many a fastidiously organised person; you either love them, or hate them. If nothing else can be said of them, they're at least a cathartic experience for some people.

Well, in this activity we're going to get you to create one of your own. Find yourself a blank piece of paper and spend the next 10 minutes constructing a list of what you need to achieve by the end of the week.

Done? What have you got on there? Does it include tomorrow's washing up? How about getting your food shopping or going out with those friends at the end of the week? No? Okay, spend a little more time adding all of those sorts of things to the list too.

Commentary:

Now, looking down that list, have you honestly got enough time to get all of those things done this week? How many weeks do you need to delay tasks to be done next week, sometime, never…? Which are the tasks most likely to slip?

Since trying to fit the most in when time is at its scarcest is rarely the most efficient way of working, highly successful executives take enormous care of time. As Drucker observes, far from taking time for granted and then trying to complete everything in a rush when a deadline looms, such executives appreciate that all time is valuable and must be used carefully, not filled with valueless activities.

Why are we telling you this? Because we would argue that despite the much-trumpeted importance of delivering on a certain delivery date (preferably as soon as possible), many of us fail to appreciate the reality of time as a resource. Instead, we behave exactly as if we were writing a bucket list – focusing intense pressure on the most pressured section of activity, when the real gains are to be made somewhere quite different – the time before the diagnosis, all those bored evenings in front of the television or wasted days in a job we knew we hated…

1.1. Delivering on time is always good

'Delivering on time is always good', a statement that rings so true as a rule of thumb that even if you are reading this with a sceptical eye, expecting an attack on it, there is probably a little voice inside you saying, 'yes but it IS good'. After all, is there anything more irritating than waiting for a train or a bus and seeing the announcement board flash up 'delayed'? When we're promised a time, we get very frustrated if it's missed, but – and here's the important bit – we want that 'promise'. Western travellers in countries such as Guatemala or Honduras find the bus systems there hard to deal with: buses leave when they are full, which might be before the scheduled time or several hours after.

Figure 5. Guatemalan bus travel

In our working lives we make promises to customers and then build internal commitments to other departments from this – perhaps your marketing department books an ad slot, or maybe the sales team hire a stall at a trade fair. All of these are real commitments. Just like travellers sitting on a stuffy bus, the customers, marketing department or sales team will get annoyed if the promised date is not met. Last but not least, a host of reporting measures assesses our performance on 'percentage of orders filled on time', 'average lead time' and other such metrics.

"The project is completed on time, and on budget, offering all features and functions as initially specified."

The Standish Report, The Key Measure of Success

The highly influential Standish Report drew on a comparison between bridge building and software projects to conclude that because design was not 'frozen', software projects were more likely to fail. Further, because their failure was less public than a bridge collapsing or going over budget, mistakes were less likely to be recognised, analysed and learned from. Despite recognising these differences, the Standish Report felt software needed to strive in a similar direction, and proposed the full definition of success as given above. 'On time' has become one of the most important measures to report on or to ask about.

Not that bridges have it all their own way. Many of you will remember the publicity that London's Millennium Bridge received; it was delivered £2.2 million over budget, and two months late. Of course, the publicity peaked when it was announced that the bridge would need to close for almost two years while work was undertaken to cure it of its wobbles. People – the public, journalists, senior managers, investors – are remarkably unforgiving when it comes to being late.

Figure 6. London's 'wobbly' Millennium Bridge

The influence of this rule of thumb should not be underestimated. Many companies have clauses inserted into contracts which punish a supplier for not delivering on time. In the case of government IT projects, this can mean significant fines from authorities for every day a project runs late. Individuals may well have remuneration tied to the percentage of projects delivered on time. As the measure embeds in an organisation, behaviours begin to change.

What would you expect to be the consequence?

Success! More projects are delivered on time!

And what does this 'success' actually mean? If you've ever been involved in creating such a schedule, you know exactly what it means. You come up with a delivery date that you are certain you can hit. This doesn't mean a date you are pretty sure you can hit if it all goes well. You wouldn't do that because you will be held to this date, measured on it, paid for it. So you want to be 100% certain. And that means you build in a buffer. Instead of offering delivery in 6 months with a 50% probability of success, you promise delivery in a year, with a 100% guarantee of hitting it. You have simply changed a '6 month delay' into a '6 month buffer'. And here's the thing about buffers – they typically get used.

Activity 2: Can you make it on time?

This activity should take you approximately 10 minutes. You can do it on your own or ask some colleagues to join you.

You are a boat builder preparing to launch a new model. You have worked with your designer and have already received construction plans for the new boat. Now you have to build it. Your first step will be to estimate how long it will take for you to build the boat. Naturally your customer wants the boat as soon as possible, and you want to ensure that you have enough time to meet the commitment that you make.

As soon as you commit to the estimate your customer will arrange the collection of the boat – so you really can't afford to be late.

Objective: Build a boat for your customer, meeting the agreed estimate.

Preparation: You will need:

- Instructions for folding the origami boat (available on the valueflowquality.com website).
- Paper for folding.
- A timer.

Rules:

- At the beginning of the game you have 50 points.
- Every 30 seconds that you add to your estimate will cost you 5 points.
- If the boat is ready early – you will have to wait for the customer to collect at the agreed time.
- If the boat is not ready for collection on time – you lose all your points, and have to do 20 press-ups for missing the deadline.

When working out your estimate keep in mind that the best people can do this in less than 1 minute; the average person takes about 3 minutes; and people who have not practiced before can usually build a boat within 6 minutes.

Play:

1. Start by evaluating the boat design.
2. Estimate the amount of time to build it. Write your estimate down – once you do, it cannot be changed.
3. Start the timer.
4. Fold your boat.
5. Wait for it to be collected.
6. Calculate your score using the rules above.

Commentary:

So – did you have to do the press ups? Most people wouldn't. The penalty for late delivery was too much to risk. In most cases, you will have set yourself a time in which you absolutely knew you would be able to build the boat.

If you finished early, what did you do whilst waiting for delivery?

In software development we very often see that when a project is set up with explicit on time metrics our estimates grow. When we know we have to hit a deadline, we add buffers and contingencies to plan for the unforeseen. The more important the deadline, the more time we tend to add. If we finish early we tend to spend the remaining time on low-value activities or simply waste it.

What if you could negotiate the delivery time of your boat with the customer after you started building it?

Imagine that you need to get your mum to the airport in time to catch her flight home. She's a worrier, and traffic can be heavy, so you leave plenty of time, and then add an extra half hour just to play safe. You arrive early – but there's no benefit in being early (the flight cannot take off early since everybody is working to that schedule), so you fill the car with petrol (which you needed to do at some point), and you have a coffee, and before you know it the time has gone.

Figure 7. Having a coffee at the airport

In business a similar thing happens, buffers typically get filled with low-value activities, because a business becomes obsessed with the need to deliver in a certain window rather than 'as soon as possible'. When all systems are built around this narrow definition of on time, it becomes very hard to deliver faster.

Sometimes the pressure for on time delivery means working unfeasible hours week after week. Occasionally it means massive budget overspend as you hire more people to do the work. But most commonly, it means that those setting the schedule have built secret buffers into the plan, which pad it out. And because others know there's padding in there, senior management ask for 'an aggressive' schedule, or to 'crash' the schedule. But because those setting the date and being measured against it know how often things go wrong, they stick the padding back in somewhere else.

It is, dare we say it, rather a waste of time in itself.

Activity 3: Who determines whether you’re on time?

In this activity, we’d like you to determine how the date by which your project must be delivered has been derived.

The approach that we’d suggest you take for this activity is to interview the leaders on the project, starting with simple questions like ’When do we have to deliver this project by?’ and moving on to more direct ones such as ‘What’s the significance of that date? What would it cost this organisation if we were late? What advantage would there be to us if we were to deliver early?’

What has this activity told you about the definition of on time for your project?

1.2. What does ‘on time’ mean?

In the introduction, we mentioned the successful manager who focuses on saving his time for valuable activities. How do we know what is a valuable activity and what is less valuable?

We ask ourselves what the customer wants.

So how do we know what value to assign to time? We ask ourselves what the customer wants.

When you order several books at Amazon, you answer a couple of questions designed to establish whether you are more sensitive to time or money: would you like your order grouped into fewer parcels or would you prefer books to be dispatched as they become available?

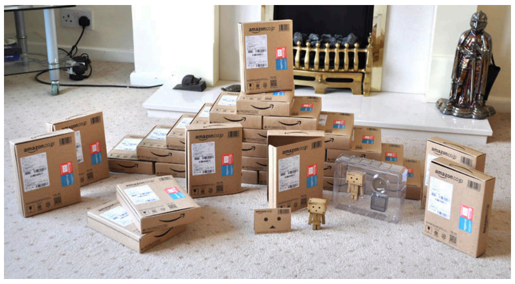


Figure 8. Lots of Amazon deliveries

If you want next day delivery then you can pay for the privilege. If you choose Super Saver Delivery (free) and you have opted for fewer parcels, then Amazon may pause the whole order until one book comes back into stock. Note that Amazon does not just do this automatically, and thus save itself delivery costs. As an online retailer, Amazon knows that if you have a specific need, you may walk into a bookshop, buy the title and then cancel the rest of your order as well.

Amazon shows an acute sensitivity to trying to find out what each individual customer’s definition of ‘on time’ is likely to be.

Absolutely on time

Sometimes the customer's idea of on time is set in stone. If you are producing a novelty Christmas animation, it needs to be ready for the beginning of December, but being ready in November is fine too. If you are an aeroplane then being late annoys your customers, but being early means the control tower will ask you to fly around until your slot becomes available.

Shades of grey

In most cases, organisations have little idea of what a schedule actually means to the customer. If you ask 'when would you like the product delivered?' most customers will say 'as soon as possible', but this might cover an entire range of meanings. For some customers it means – 'urgently, and I'll pay double if you get it to me tomorrow'. For others, if offered double price for delivery tomorrow, standard price for delivery in a week, or half price for delivery in a month, they would select the lowest price.

Obviously delivering early or late against the estimated date will mean very different things to each of these customers. Your organisation probably knows the cost associated with hitting the delivery date (lots of overtime, hiring extra staff, switching people off other projects), but it is unlikely to know the true cost of delay to a project. The result of not knowing this can mean going all out to hit a deadline that doesn't matter (and hence costing the company more than the project is worth), or letting a deadline slip and losing the customer.

In the absence of real measures, most organisations fall back to on time delivery as their key performance indicator. They base on time delivery not on the customer's external view and the impact on the business, but on an internal deadline; a date normally estimated at the beginning of the project, perhaps even prior to any real understanding of scope. Since most organisations have no idea of the customer's view of the date, on time delivery in fact gives nothing more than an illusion of control.

CASE STUDY: A Titanic delay

James Cameron is a famous film director; Terminator, Aliens, Titanic and Avatar are just a few examples of the movies under his belt. Without wishing to discuss the films' artistic merit or otherwise, there can be no doubt that they have been major successes, not only grossing billions at the box office, but also gaining Academy Awards.

Titanic had an initial production budget of $75 million which ballooned to $200 million (making it the most expensive film ever made at that time and 22 times as much as it cost to build the original ship in 1911). Moreover, it committed what in movie circles is the ultimate sin (far, far worse than going over budget) – it was late.

Principal photography took a month longer than scheduled, while the post-production special effects (CGI was a new art) were even more behind than filming had been. A release date of July 1997 had been announced. In May 1997, James Cameron admitted the film would not be ready despite having already spent three years in production. The studio announced a revised release date of December 1997. The media enjoyed themselves trashing the big budget, overdue film.

And yet when the film finally did open, it received positive reviews. Eventually the film earned $1.8 billion worldwide and picked up 11 Academy Awards.

James Cameron and the studio had invested so much in the film that by the end point, it was far more important to get it right, than to cut corners to be on time.

It is possible that the studio had no other choice but to continue the project. Yet with film after film, James Cameron has repeatedly placed more emphasis on getting the film right rather than delivering on time or on budget.

Figure 9. James Cameron collecting his Oscar

1.3. Focusing on the outcome

Activity 3: On time and on budget - wouldn't that be nice?

We've a little experiment for you to run in this activity. Find yourself two index cards, or similar sized pieces of paper. On the first, write 'On Time' and on the second, write 'On Budget'.

Now, take these two cards and find the leaders on your project. Ask them the question 'If you had to choose one of these for this project, which would it be?'

How easy was it for the person to decide on which answer to give? Did any of the people you asked suggest that they couldn't decide between the two?

Ask anyone (a customer, your boss, the cleaner) whether they would like things faster or cheaper. Most will answer promptly, 'BOTH'. That's very human. It's also not realistic. Yet just as with book delivery options from Amazon, the fact is that people will make choices that compromise between the two depending on the outcome they most want. It's not only that we need to ask when customers want a product or service, but that we should consider what it will cost us to meet their expectation or even what we might gain by delay.

Speed by itself is not the goal

Actually, if you are a Ferrari Formula One engineer, then speed is the goal. The expense of a new wheel design is completely irrelevant to them. The fact that your new wheel stays at top performance for longer but may be more likely to get damaged, however, is a very important trade-off. A great deal of effort will go into the decision over whether to adopt the new wheel design or not.

For engineers working on normal cars, the design requires balancing competing needs: speed, fuel consumption, cost of manufacture and many other considerations. Some customers will care more about speed than fuel consumption; others are most sensitive to price. The point for those car engineers, just as for most of us, is to make a product that our customers want.

As we pointed out in the introduction, delivering fast is an advantage towards that goal, but it is not the goal itself. In Developing Products in Half the Time, Smith and Reinertsen point to Black and Decker as a company that eventually decided it was innovating too quickly, that the associated costs were too high and that they wanted to space out innovations to allow each product a natural lifecycle in the market. Although as the authors drily note, "most companies are far from that point."

CASE STUDY: Salon.com

The world's first (or so it claims) entirely on-line commercial publication, salon.com, was founded in 1995 providing free content on a range of subjects from politics to entertainment. By 2003, the company had accumulated so many losses, and was having so much difficulty paying its rent, that it needed to ask readers for donations to keep going.

The business took some painful measures (cost-cutting and redundancies) and tried to post lots of content in order to raise their search engine ratings on important topics and therefore bring new readers to the site. Given few staff had survived the cost-cutting exercise, the quality of the content inevitably went down.

By the end of 2011, things were looking much better for the company, hitting 7 million unique users for the first time, and managing to hit it again (and even increase the number to 7.23 million) in January 2012. What happened? Well, the editor-in chief went against the received wisdom of posting lots of content frequently in order to raise the company's ranking on the search engine. Instead they slowed down their process. Writers took longer over each article, they ignored quick stories which could be found elsewhere, they reduced the number of articles, publishing one-third fewer stories on average than they were the year before. The result was that traffic to the website increased by 40%.

And the moral of the story? Speed was not the answer for salon.com; original content was what their readers wanted, and they were prepared to wait for a better quality story from salon.com than to get a quick rehash of what other news sources were providing.

1.4. Identifying the desired outcome

In the James Cameron case study discussed previously, a great deal had been invested in the July 1st film release date: marketing materials, cinemas, press… changing the deadline was painful. But Cameron was focused on the true outcome – making a film people would love.

When managers say they would like to work twice as fast, what they often mean is that they would like to get twice as much done using the same resources. This is a confusion of two ideas: speed is not the same as productivity.

If you want me to get from A to B fast, I will drive my Ferrari.

If you want me to take 10 people from A to B, I will hire a bus.

Now, I could drive my Ferrari from A to B 10 times, taking a passenger with me each time, and back again 9 times to pick another up. It is just possible that this would be faster than the bus. But you might find the cost – or the risk (I'm probably going to break the speed limit) – too high.

Figure 10. A 'Ferrari' Bus

This may seem obvious to you in the abstract example given above, but the truth is that many managers say they want to deliver at speed, when all the actions they take suggest that what they really wish to maximise is throughput. Delivering at speed has associated costs and risks. Focusing on the customer and on the desired outcome will help you decide how you actually want to travel.

Let's examine different customer needs or situations and their impacts on a business response.

Situation 1 – Customer will wait, however annoyed she gets

A patient makes an appointment to see her doctor. She may spend quite a long time waiting in a telephone queue in order to make the appointment in the first place. Then when she turns up at the surgery she will wait until the doctor is ready to see her (often behind schedule). The doctor decides that some further tests need to be done and refers her to a specialist: an appointment comes through for two weeks time and once again the patient turns up to wait, the tests are done and sent off to the lab. A week later the patient returns to get the results…

Figure 11. Waiting to see UK NHS doctors

You get the picture. The actual treatment or testing time is very small – probably 20 minutes, but the process takes a long time. The patient is prepared to wait. She knows a great deal depends upon the outcome of her diagnosis, so she is unlikely to give up, however frustrated she may get. That enables the doctor's surgery and hospital to focus on efficiency.

The doctor's view of his surgery will be a very different one to the patient's. He is seeing a different patient every 5-8 minutes. His throughput is high, as is the hospital's and the laboratory's. The individual patient's experience is considered unimportant compared to the goal of maximum efficiency for the surgery and hospital. Note that this may not equate to maximum value for the system as a whole – if patients cannot get to see their GP, for example, they may go to the Accident and Emergency Room at the hospital, causing long delays in a more specialist (and expensive) part of the system.

Situation 2 – Customer will get frustrated; may walk away

Now compare this to a supermarket. Waiting in a queue at the checkout annoys customers. They get bored, their children cry, and the experience gives a disproportionate feeling of negativity towards the store. If the queue is sufficiently long and slow, they may give up altogether – actually abandon their shopping and walk away. Because this is a real risk, supermarkets have flexible numbers of staff, extra people who can be switched onto the checkouts at busy periods to keep queues down. They monitor and try to predict queues – both in obvious ways (lunchtimes and after work are busier than 11am), and through more sophisticated footfall analysis – if a certain number of people have come through the door, extra staff are called to man the checkouts.

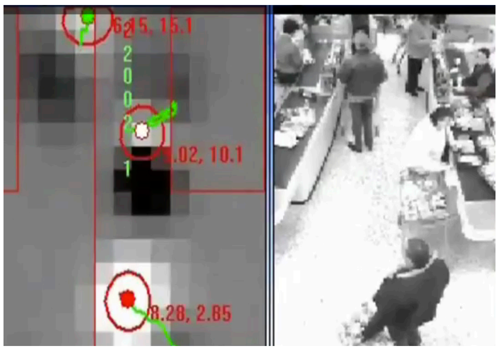


Figure 12. Tesco using thermal imaging cameras to determine queue lengths

Tesco introduced its 'one in front' initiative: if there is more than one person in front of any given customer in a queue they will open another checkout. This is obviously a more costly system – a higher proportion of staff must be trained to use the checkouts, extra staff need to be available, managers or supervisors need to monitor the queues constantly… The customer's need – speedy paying – demands flexibility from the store.

Activity 5: Decisions, decisions...

In Activity 3, we asked you to identify how the delivery date for your project was decided. Then in Activity 4, we highlighted the point that ‘On Time’ and ‘On Budget’ are two separate concepts, even though many people find it hard to choose between them and are convinced they can have both.

Depending upon which choice you made for your project from Activity 4, in this activity we’d like you to consider what effect it would have if this knowledge were widespread. Would it impact on daily operations, for example? How might decisions over scope differ?

Delivery on time

Often, as projects near their end, we reduce their scope as we realise that they are not going to deliver on time. In many cases this is too late. If you have decided that your project absolutely must be delivered on time, then create a list of which features could be dropped, beginning with the least important and stretching back until you reach ‘core’ functions without which the project cannot be considered a success.

Delivery on budget

When cost is the most important issue, projects tend to integrate with existing features so as to keep costs down. Rather than building all of the functionality they need, they build on top of existing functionality to release a version of their software with less effort. Sometimes this means employing workarounds or accepting less than perfect functionality. Is this a strategy that could be used on your project? Identify the functions that could still be employed.

Build a new rule of thumb

Based on your analysis, what simple decision rule - a rule of thumb, perhaps - could your project use to help make decisions in daily operations? Discuss the idea with colleagues. If you feel that it would benefit your project why not work to see it made operational?

2 HOW MUCH DOES DELAY MATTER?

If a product is late, does it really matter? We talked in our introductory sentence about customers walking away, the risk of technology changing...

OK, so let's say it does matter. But how much? Should we hire 30 more developers to hit this deadline? Should we airfreight goods rather than shipping them by sea? Should we go with this supplier who can deliver in a week's time or should we wait a month for the supplier in China who's half the price?

To answer such questions and to balance competing objectives, we need to come up with a way of quantifying the time sensitivity of a project. We normally call this the cost of delay – and although it is no more accurate than any other numerical prediction, at least it tries to make our assumptions concrete – allowing us to test them.

If we are unable to do this then we end up relying on internal rules of thumb (the ones we said in the Why Change session were wrong) or decisions that are influenced by our own pet preferences. In the diagram below, we show that not adding 30 developers may mean late entry into the market. We compare the cost of the delay to the cost of the additional developers.

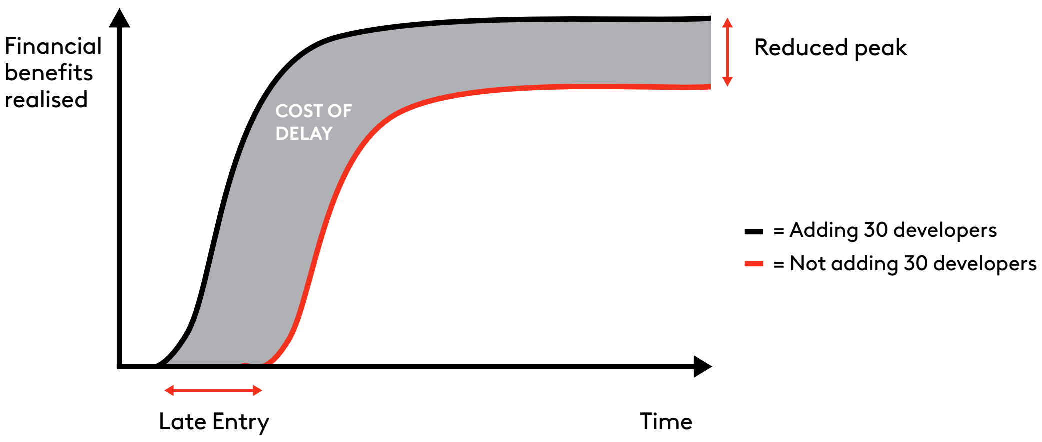


Figure 13. Example cost of delay chart

"Each product has a certain cost of delay that will determine how much effort should be put into accelerating it."

Don Reinertsen

Working out the cost of delay is not always the simplest calculation, but it should be done at the beginning of each project as part of the business case. It simply asks us to calculate the total business value that we expect to realise from a given product over a given period (usually 3 years). Then divide the total value to give a monthly or weekly cost of delay.

This extended piece of work is not necessary for everything – it comes into play where decisions are significant or decisions are marginal. We don't always need to do a calculation at all. We can ask ourselves questions to help give an indication of whether the cost of delay will be high, medium or low: are customers likely to go elsewhere? Is there lots of competition? How important is this to our business' core strategy? What won't happen if we launch late? Is there a seasonal aspect linked to our sales?

Make no mistake – however rough your estimate may be, the cost of delay is a real cost. It can feel very hard for managers to accept its importance because the cost of delay is not money that we see draining out of the company bank account week after week, or out of our departmental budget, but it is the cost of opportunity.

Businesses are about creating value for customers, and the cost of delay measure is an important one because it represents customers and their value to the business. Ignoring it can lead to the failure of your company just as easily as uncontrolled spending.

Activity 6: What's the cost of delay?

Most project business cases contain information about their expected return on investment (ROI). That information can be used to generate an approximation of the cost of delay.

In this activity, which will take approximately an hour, we'll review the business case for one of the projects in your environment, estimate the cost of delay for it, and look to deploy it as part of the decision making toolset.

Start by reviewing the business case for your project, noting down any of the suggested returns. If you encounter returns that are spread over a time period (IT projects commonly refer to returns over a 3 to 5 year period) simply total those in to one number – there are circumstances which would mean that when the returns appear would have an affect of decisions using cost of delay, but in the interests of simplicity, we will ignore those for this activity. If you need to make some assumptions – that's fine too, we'd suggest you note them down, when you share your calculations with others later in the exercise they maybe able to improve your data if they know your assumptions.

Okay, by now you should have a set of numbers that you have taken from the business case. The next step is to total these numbers together and divide that number by 12 – we're interested in a monthly delay cost.

Similar to the question we posed in Activity 5, we'd now like you to think more about how this knowledge could be used in your project. In discussions about whether to attempt to reduce costs by taking a slower approach, would the cost of delay influence decisions otherwise?

As a conclusion to this activity, we'd like you to take this calculation and show it to a few other people involved in the project to see if they agree with how you've arrived at your estimate – could it be improved with their help?

Once you've done that, we challenge you to take your cost of delay calculation to those people on your project who you feel would benefit from it, and help them understand how they could use it.

2.1. Cost of delay and capacity

When considering how we could deliver a project faster, most of us think about whether we can afford to add extra capacity. Like Tesco opening tills to reduce queues at the checkout, we wonder if we can afford more people – more developers, more testers, more contractors.

Sometimes hiring more capacity is the right decision, justified by the cost of delay if the project is late. But it is not always so easy. Recruitment is often a slow process. You need to gain approval to add to head count or budget, you need to advertise, interview and then – even if you have found the right person – you need to get them up to speed on the project, maybe even train them in your particular system. By the time this is done, either the crisis has passed or the project is even later and now you need even more extra people to get it back on track!

Even worse, as we go on to discuss in the Teams and Motivation sessions, adding extra people doesn't always help. In fact, increasing team size can slow a project down. While forcing everyone to stay late or paying overtime can be counter-productive as well. So what do we do?

Most organisations need to think about how they manage the capacity they have better. How can they optimise their flow to free up time? How can they make their capacity flexible: that is, just like Tesco having staff who can man tills, stack shelves and reorder stock, how can development departments ensure staff can design, build, test and organise their own work? How can they eliminate any waste that clogs up the system? How can they set themselves up so that structures support fast development rather than obstructing it?

We'll go on to look at the answers to these questions, but first a word of warning. Doing all this isn't easier than simply adding capacity – it's just more effective. It may be cheaper in the long-run, but it doesn't always look 'efficient' in the short-term.

And finally – going faster is usually achieved by having excess capacity in the system already.

What? You tricked us!

Not really. You don't need to hire anyone new or pay out masses of overtime, but you do need to ensure your people aren't so overloaded that there is no spare capacity (no Slack, as Tom DeMarco called it) for them to handle the changes and uncertainty that come hand-in-hand with faster development.

> "This is perhaps the hardest concept to get across to managers who have been indoctrinated through years of experience. In fact, excess capacity is critical to making a development process work. If you fail to have excess capacity you will pay the price in cycle time."
>
> **Don Reinertsen**

Tom DeMarco used an excellent example to explain the concept. Remember the little number puzzles where you slide tiles around (sometimes known as the mystic square)? These puzzles rely on having one space free in order to move the tiles around. Fill all the spaces up and you can't move anything – which is fine if you begin with the numbers in the right place (rubbish puzzle that would be, mind you), but hopeless if you need to adjust anything.

Activity 7: The slide puzzle

If you've visited one of the many web based slide puzzles on the internet you've probably already wasted too much time on them. If you haven't or if you've never come across this version, then try the train ones here – http://www.tilepuzzles.com/default.asp?p=18

The train puzzles mean that you need to move blockages around in order to free up a route (sound a bit like flow?). The more blockages the harder the puzzle is.

When running a project at work, you often need to move people or hardware around to free up capacity, or juggle critical activities to ensure the flow of work continues. Of course if you can find time to get everyone playing the train game then it suggests you have a good amount of spare capacity. We recommend setting up a league.

3 FLOW

3.1. Optimise the whole, not the parts

In work, as in life, it's easy to get hung up on the detail. Ever worked with someone who is convinced that his tiny bit of the process is the most important of all? Other people can have their budgets cut or their schedules changed, but not his. His is special. And sadly, it's not just one difficult guy, it's all of us. It's part of human nature to get deeply involved in our own specialism and to devote our time and energy to making that little section the best it possibly can be.

As a problem, it's so universal that it has spawned comparable proverbs in many different languages. Americans lament what happens when, 'you can't see the forest for the trees', the Brits (just to be different), say things are difficult when 'you can't see the wood for the trees', while the French talk about 'the tree that hides the forest'. In business we have rather less appealing idioms. We would say that it's essential to consider the primary purpose of the whole system: delivering value to the customer. And that means that optimising any one process within the system, or even all the processes individually, can never be as effective as optimising the whole.

Figure 14. Can't see the wood for the trees? (Created by veneerselection.com)

Activity 8: Pass the parcel

Objective: Pass the ball as many times as possible in the allotted time.

Preparation:

- A ball (though a screwed up piece of paper will suffice).
- A timer.

Play:

1. The activity is played over 3 rounds.
   a. Ensure that everyone is spread far enough apart to make it easy to throw the ball between each other.
   b. Each round is time boxed to 30 seconds.

2. **Round 1**
    a. Pass the ball around the group as fast as possible in the allotted time. Record the amount of passes you achieved.
    b. The participants should take 30 seconds to consider how they might improve. They must do this in silence.
3. **Round 2**
    a. Without discussing the improvements you intend to make with the other people in the group, form the group again ensuring you have enough space between each person to easily throw the ball.
    b. Pass the ball around the group as fast as possible and at the end record the amount of passes achieved.
    c. At the end of the round the participants may this time participate in a group discussion on how to improve for up to 30 seconds.
4. **Round 3**
    a. Form your group again and throw the ball between yourselves as quickly as possible for the 30 seconds and when complete, record the amount of passes.

Commentary:

How did the amount of passes differ between Rounds One, Two and Three? We typically see an improvement between each and sure there's the possibility that you were all just getting better, but there are normally some significant steps upwards at each point. How about the improvements that were made, how did they differ between Rounds Two and Three? We've seen some incredibly innovative improvements made when people have worked together, one group, for example, excluded all but 2 participants from throwing the ball between themselves and stood a matter of centimetres away from each other, meaning that they achieved a massive score.

Imagine an aerosol deodorant factory: empty cans are loaded onto the line, filled with a squirt of perfume, a valve dropped in, crimped to make a tight seal and then filled with gas to the correct pressure. A cap on top and the aerosol is ready to be packed into a box.

An engineer has invested months of his time in making the machine that crimps the valve onto the can work slightly faster. He feels extremely pleased with himself, but to his surprise no-one else is. In order for his work on the crimping machine to make any improvement, more cans need to arrive there. That means people need to place cans on to the line faster. The pressurised filling machine also needs to be able to fill the cans more quickly – otherwise half-finished aerosol cans will begin to pile up in a heap. These pose a safety risk. The 'efficiency' that the engineer has achieved is not only a waste of time since it's not delivering finished goods out of the door any quicker, but has now become a safety hazard!

Figure 15. A valve crimping machine in action

The smooth progress of work throughout the entire system is what we refer to as flow. Anything that halts or disrupts the work is a block or a delay – the enemies of flow. If the machine that created the crimped seal had been much slower than the other machines or processes on the manufacturing line, then we would have seen cans piling up in front of it (a queue). This would have shown us that the process was a bottleneck in the system's flow, and investing time and money in speeding it up might have been a good idea. Note that we say might... If cans ready to be crimped are safe, and cans after crimping are dangerous, then we might want to adapt our flow to make sure we never have a queue of pressurised cans, even if that means that the throughput is lower or the speed is slower. Flow is not about all out speed – it is about processing work in a way that balances the company's competing needs and requirements in order to deliver the best final value to the customer. After all, if the factory blows up because of an aerosol explosion, no-one is going to get much value.

"We increase return on investment by making continuous flow of value our focus."

Declaration of Interdependence, a document on Agile Values

Functional departments need to be just as aware of the overall flow of value as those handling inflammable substances. Jim Highsmith and co-creators of the PM Declaration of Interdependence placed it number one in their values. We look at the entire flow because by doing so we are able to deliver the most value in the shortest amount of time. We'll continue to stress (until you're repeating it in your sleep), **value** is the primary goal. Speed is simply a means to deliver value. How fast we need to go must be judged by how speed affects the value. But however fast we want to go, we should look at the whole process and turn our attention to the areas that constrain the flow or where there is unnecessary delay.

This concept embraces the extent to which different parts of the system impact on one another. Jim Womack, Chair of the Lean Institute, quotes a particularly bizarre example of short-term and partial thinking by an American car company. This company announced a special incentive for sales people to sell cars in order to run down inventory. They forgot to call a halt to other parts of the system's workings, however. A large number of sales led to an increase in production as work was 'pulled' through the system (in the approved Lean manner). This of course led straight to – increased inventory!

An idea intended to reduce waste actually ended up increasing it, because the company had not considered the whole system and its flow.

The anecdote explains three terms you may come across in Lean thinking: muda (waste), mura (uneven demand) and muri (overburdening the system). The terms themselves aren't important, but the concept of how they are related is. In our aerosol can example, by increasing production at one point, the engineer created uneven demand and overburdened the next part in the process. Doing so led to a dangerous amount of inventory – which is waste. He did all this while thinking he was being efficient.

Uneven demand and overburdening the system can either be seen as different types of waste or as activities that will lead to waste. That's why, in the section that follows in which we look at ways to reduce waste, we want to be absolutely clear that this should never be done without thinking about the system and its flow as a whole.

There's nothing more infuriating than initiatives intended to make improvements that actually have unintended negative consequences.

3.2. Finding waste

Have you ever gone on holiday with friends or family and had to wait for them to get ready to leave? Maddening, aren't they? Especially if you're in a hurry. You can cope with them rushing back to fetch passports or forgotten wallets – those are important activities. You can even manage to handle it when Great Aunt Susie decides she'll just pop back for the lavatory. It's all the other stuff… the whatever-on-earth-are-you-actually-doing? Fussing. Fiddling. Faffing.

Figure 16. Not all activity is valuable (graphic by fuoro)

Getting a project out on time can have rather the same effect. The fact is that not all activity is valuable. A group of people can rush around making lots of noise and using up masses of energy without actually producing something valuable. As Rene Domingo wrote, 'An output acquires value if and only if it is taken in – used or bought – as an input by another process - another worker, machine, or customer.'

A big process review may take weeks to carry out, write up and present, but the vast file of paper that results from this does not, in itself, represent value. It is only if the process review assists a company to respond to customer demand that it is valuable. Otherwise, it is simply getting in the way. Don Reinertsen tells an anecdote about visiting a company where the project had a $3 million cost of delay per month. The CEO also visited once a month and the team were required to stop work for a week to prepare a project review for him. In Lean manufacturing, such activity is described as 'waste'.

If we eliminate the faffing time, we can get on the road much faster; if we eliminate the waste in our organisations, we can often deliver our projects much earlier. If the company above had stopped doing the project reviews, they would have gained 25% extra capacity. There's no need to hire more people, no need to force everyone to work weekends, no need to introduce brilliant new initiatives, we can just get rid of the waste. Who wouldn't find that attractive as an idea? And the difference can sometimes be astonishingly dramatic.

We're going to consider types of waste or valueless activity that halt us from getting more valuable things done. However, it's important to remember that no matter how hard we concentrate on eliminating wasteful activities, we should always be aware that the biggest waste of all is to work on the wrong thing. It is irrelevant that you're efficient, if you're producing nothing of value, and although producing the wrong thing fast and cheaply is certainly better than producing it slowly and expensively, the wrong thing is still waste! That is why thinking about how we optimise flow or eliminate waste is only part of the answer. It must go hand in hand with delivering value early and often to gain faster feedback.

3.3. The Lean view of waste

The Toyota Production System was famous for its focus on eliminating waste as a method of raising productivity. Taiichi Ohno, who was one of the originators of the Toyota philosophy, demanded a very strict definition of the term. Waste did not mean only reducing the number of damaged parts that might have to be thrown away (traditionally referred to as wastage), waste incorporated everything that was not creating value for the customer.

As the ideas were tested and proved in the factory, the system was rolled back to include waste in product development as well. Consider how much work went into design and engineering – Toyota realised this effort did not deliver value, until the car rolled off the production line for a customer to drive away. That meant waste in the up-front work should also be ruthlessly hunted down and eliminated!

In IT, service and product development, this means looking at the entire development process, finding the huge savings that can be made – not only while actually producing the working product but earlier in the product lifecycle, in the research and approval phase, for example. This is the area that Reinertsen calls the 'fuzzy front end', where dramatic gains can be made.

A common mistake is to concentrate on delays in the area where time is already tight: development itself. As a deadline looms we all begin to look anxious. People stay late; pressure builds; slips and mistakes become more common. As Tom DeMarco comments in his book Slack, 'People under time pressure don't think faster…' Since most development is by its nature creative, thinking is exactly what people need to be doing. Removing people's thinking time normally leads to shoddy work and another of those forms of waste: defects.

Instead, we need to look for wastes and delays that are blocking the flow more widely, across the whole value chain. In Implementing Lean Software Development, Mary and Tom Poppendieck described the full schedule as reaching, 'from the moment the customer gives us an order to the point when we collect the cash'. Value flow encompasses all the processes that exist before code is written or an order is received, as well as those that exist once the final test has been run but before the money starts rolling in. Actually, it doesn't stop there, since the way we interact with our customers after they've bought from us is also just as important.

Mary Poppendieck has helpfully suggested how the manufacturing list can be translated for software and product development to show the types of waste that occur there.

Seven Wastes of Manufacturing	Seven Wastes of Software Development
In Process Inventory	Partially done work
Over-Production	Extra Features
Extra Processing	Relearning
Transportation	Hand-offs
Motion	Task Switching
Waiting	Delays
Defects	Defects

"Waste is code that is more complex than it needs to be. Waste occurs when defects are created. Waste is non-value-added effort required to create a product. Wherever there is waste, the Lean practitioner looks to the system to see how to eliminate it because it is likely that an error will continue to repeat itself, in one form or another, until we fix the system that contributed to it."

Alan Shalloway, Lean-Agile Software Development

Activity 9: The waste snake

This activity requires the whole team and is done as part of your normal work.

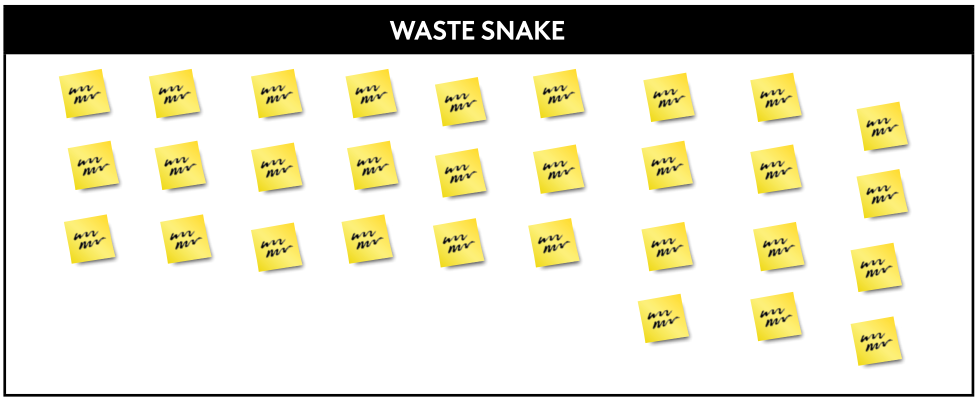
Set up a board (or sheet of flip-chart paper) on the wall with a pack of Post-it notes beside it. Every time someone does something they consider wasteful, they should write it up and stick it on the board.

Each note should include:

- A short description of the activity
- The length of time the activity takes
- The initials of the person recording the waste

Here are some examples of things you might include:

- Waiting for the tests to run (14 minutes!)
- Annabel not available to provide feedback on feature – 3 days
- Joe was off and he's the only one that knows about RabbitMQ, so I couldn't progress my feature (7 days)
- The dev-test servers were down for 2 days while updates were carried out – we couldn't deploy



Use Poppendieck's table when thinking about what types of waste you are looking for – you will probably find examples of every type. You can also group them by activity type – for example: 'support', 'release' or 'production defects'.

By linking each one together the team build a 'snake' that forms a visual representation of the amount of waste the team is carrying.

This can form the basis for a discussion (during Retrospectives, perhaps) of which wasteful activities the team will attempt to eliminate. Sometimes these will require investment – by building a picture of the length of time the waste takes (or how often the sticky appears on the snake), the team can build a case for the investment.

3.4. Eliminating waste is not the same as reducing slack

Before we delve as excitedly into waste as Pixar's WALL·E, let's take a moment to remember why we care. We only eliminate waste when we believe it will help us deliver more value.

If we can produce more value by ignoring waste, then we shall do so. Imagine the kind of manager who chases workers to ensure they have not used more than a regulation number of paperclips, or who demands that everyone limit visits to the bathroom to five minutes, three times a day… He might think that he is reducing waste – but actually in wasting his own time and everyone else's, he is a prime example of the wider meaning of waste – activity not focused on customer value.

In fairness, managers are rarely this insane… but the passion for efficiency can lead us to dangerously overload capacity. You may hear this concept described as 'muri', which literally means unreasonable, but represents unreasonable demand or overburdening a system, often caused by focusing on just a single point.

We remind you that overloading the system with demand and forcing everyone to be constantly busy, does not provide the highest throughput or flow of value. It does the opposite. It causes congestion and delays, and hence reduces value. The higher your capacity utilisation, the longer your queues (think of the patients waiting in the doctor's surgery); the longer your queues, the more time each project spends delayed rather than being actively worked on.

These issues lead to two of the types of waste on Poppendieck's list: partially done work and delays. Tom DeMarco called it: 'a dangerous corporate delusion: the idea that organisations are effective only to the extent that all their workers are totally and eternally busy.'

The moral of the story is – do not confuse waste and slack, they are two very different things.

3.5. Delay is the enemy of flow

"Delays represent waste—if you remove them you will deliver faster … While the benefits of delivering fast are clear, it is essential that this is done in a sustainable manner."

Alan Shalloway, Lean-Agile Software Development

We call this waiting time a 'queue'. This is where our work spends most of its time – sitting in a queue waiting for someone to do something. It can have a very serious impact on our delivery of value. Remember the patients in the NHS waiting room? What if someone comes in with chest pains? They may have a heart attack in the waiting room while sitting in the queue for the doctor! The risk of something really valuable being delayed in the queue is high – one key reason why we should invest time in prioritising our queues. The other impact is that queues slow everything down and they tend to get longer quickly.

Figure 17. Queues at airports can take up to three hours at busy times

Have you ever stood in line at an airport waiting to have your passport checked? If so you almost certainly know the experience of thinking to yourself, 'oh, I picked the wrong line!'. One person seems to be having a full blown argument about something, while the family are hunting through a thousand bags for the baby's passport…

Tasks – whether checking a passport or writing a feature – normally take longer than expected, not less time. And the effects of each delay are cumulative, meaning as the queue grows, the likelihood also grows that the queue will get even longer and the delays increase further. This can quickly spiral out of control. In the airport people will begin to re-assign themselves to different queues or staff will open another desk, but in software development if no one is looking out for the queue then it can take longer for anyone to realise there is a problem. By that point, the project could be seriously delayed.

Handling the delays caused by queues is thus of primary importance to optimising flow. Just as we made a distinction between eliminating waste and running with no slack (a problem that leads to 'muri'), so we need to distinguish between eliminating queues, and running unevenly.

Think of driving a car – it is inefficient to act as if the accelerator and the brake are the only two ways of controlling our speed that we have. Switching from accelerator to brake would cause the car to bunny hop down the street in a series of jerks. It's a wasteful way of driving (not to mention potentially dangerous!). Good driving tries to allow the car to slow naturally before gently applying the brake, anticipating things like a corner or a queue of cars ahead that might mean the driver will need to slow down.

In product development, or in manufacturing come to that, we try to act in the same way, ensuring a smooth flow of work through the system. In practice this normally means having a small amount of buffers. For example, rather than a development team finishing every piece of work and then calling a meeting with business stakeholders to decide what they should do next and break down the work, you would expect the team to have a small buffer of 'engineering ready' items in a backlog or buffer from which they could pull.

Thus, don't confuse the importance of running with very low queues with starving the development pipeline! Uneven demand causes waste of its own – you may come across it described by the Lean term 'mura', literally meaning uneven.

3.6. Partially done work

What is the main cause of queues and delays? Partially done work or – as we tend to call it throughout these session books – Work in Progress (WIP). Again, this is so important as a subject that an entire session book goes into it in more depth – read the Work in Progress session.

Having lots of tasks to do, numerous features to code or masses of projects in hand are all examples of running with high WIP. High WIP (as mentioned above) leads to long queues and long delays – all of which means we take longer than we would like to deliver our software.

Lots of partially completed work carries a heavy risk that it will cease to be valuable. That would mean that all the effort put into the work so far could end up being waste. We are very aware of this when we think about physical inventory – stock or raw ingredients sitting in our warehouse – but find it harder to realise that partially completed work is simply intellectual inventory.

Just as raw materials and stock can pass their sell-by-date, so our partially done work (half-coded systems; big piles of user stories; beautiful mood boards and customer personae) can become valueless. Perhaps the market moves on or a competitor launches… all at once we need to change everything. Or perhaps we tell ourselves that a project is 'nearly done' only to find that we have terrible integration problems and that 'nearly' actually means a further four months. Our partially done work has suddenly translated into a heavy cost and risk to which we had previously been blind.

This is the main reason why we focus on flowing individual pieces of work through the system – ensuring they are really 'done' and ready to release, and then releasing them as soon as possible.

3.7. Extra features

The Standish Group's CHAOS Reports track the usage of features in a given program. The results make for painful reading. Over 60% of features are rarely or never used. Let's just think about that staggering figure. That means 60% of the development work that went into that project was wasted. Worse, it means that all the work that goes into maintaining the product, all the complexity that more than half of the product added to the whole, is also redundant. We have poured money, effort and creativity into something that nobody wants.

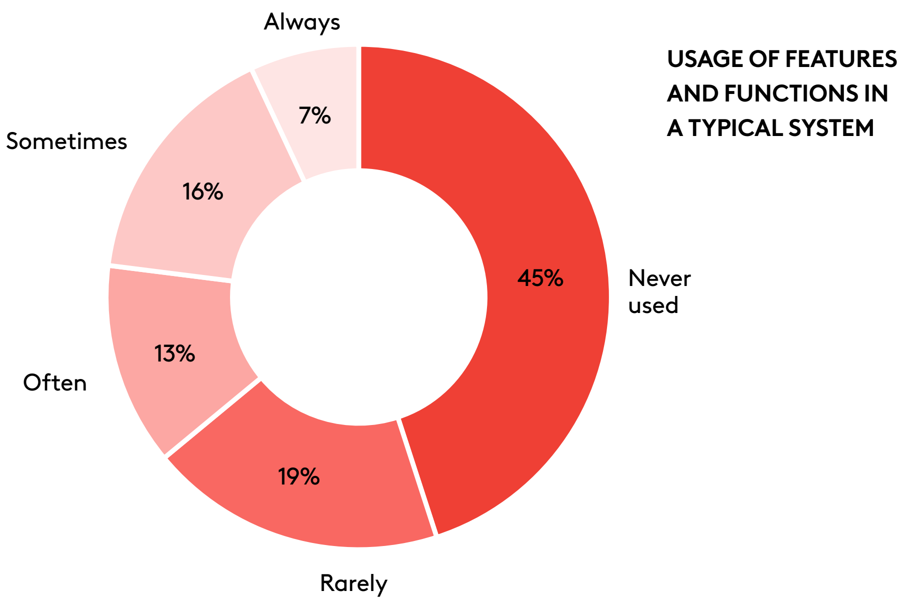


Figure 18. Standish Group study of 2000 projects at 1000 companies

Michael George in his book Conquering Complexity in Your Business, suggests that complexity is like cholesterol: It clogs up the arteries of an organisation; it is 'the silent killer of profits and growth.' That is how dangerous the need to constantly maintain unused or overly complex features can be – let alone the headache it gives when considering integration and reuse.

What's the answer? For Mary Poppendieck it's simple: write less code! Boehm uses more words, but essentially says the same thing:

> "One of the best ways of reducing software costs is to reduce the number of [features] we must develop. We can do this by using already-developed software, as with software packages, adapted software, or program generators; or we can devote more attention to avoiding the development of unnecessary software... For any software product, we can think of a long list of features which will enhance the product in various ways ... Others have usually ended up as gold-plating."
>
> **Barry Boehm,** Software Engineering Economics

So who are these idiots writing features no one wants that clog up the whole company? Umm...

That would be all of us. Because what the previous quotes does not mention is that these 'extra features' are a normal reaction to uncertainty.

We are not deliberately stuffing projects with make-work and useless features. Instead, we simply don't know what our customers will want. Or we think we know, but we turn out to be wrong.

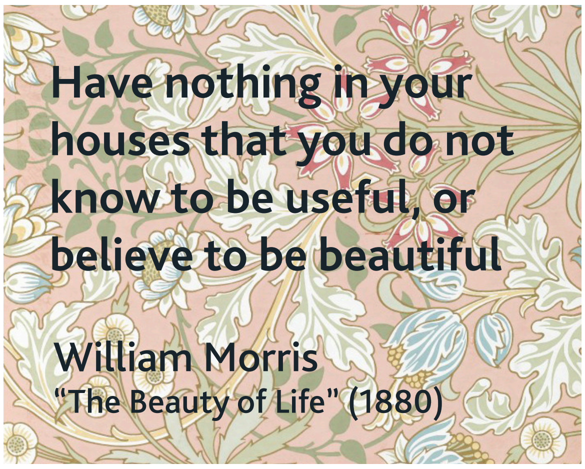
Remember our opening image of trying to shepherd your family on holiday? When your partner runs back at the last minute to fetch a pack of cards, her insistence on delaying the car could be either a typical example of how she is always infuriatingly late, or yet another proof of her brilliant foresight… If it rains every single day on holiday then the kids can improve their poker playing; if it's glorious weather then the cards will simply sit in her handbag taking up space and allowing you to make smug comments about waste.

We face the same uncertainty when creating a product. Everyone has their own ideas of what is essential and what is a nice-to-have feature. Views on which is which do not always match. Rather than arguing for days, it can feel easier to build all the features.

Poppendieck writes that we should all have the discipline to: 'Justify Every Feature: The first step to controlling complexity is to aggressively limit the features and functions that make it into the code base in the first place. Every feature that gets developed should pass the hurdle of proving that it will create more economic value than its fully loaded, lifecycle cost.'

We would argue that this still doesn't go far enough. After all, those justifications are based on assumptions and a good healthy dose of subjectivity: "I just know that customers will want an avatar where you can change her hair colour every day. That's what makes this game fun."

William Morris, the famous arts and craft designer, said: have nothing in your house that you do not know to be useful, or believe to be beautiful. We can paraphrase him for IT: have nothing in your product that you do not know to be useful, or believe to be valuable.


Have nothing in your houses that you do not know to be useful, or believe to be beautiful
William Morris
"The Beauty of Life" (1880)

How do we test our belief in what is valuable, so that we can focus on those features? The only way is to find out directly from customers what they value. Old-fashioned product development had to rely on focus groups, interviews and blind tasting tests; today in our software-based social world we try to validate our learning through prototypes and wireframes.

Most importantly, however, we try to launch something smaller earlier. The Minimum Viable Product (MVP) is a way to test our business hypotheses – not with focus groups or surveys – but with real business data. This allows the most robust understanding of a customer there is because it is not based on what they say they will buy, but on what they actually buy. We build in frequent feedback opportunities in order to guide us on what to build, what to change and what to forget about.

> "The cheapest, fastest, and most reliable components of a computer system are those that aren't there."
>
> **Gordon Bell**

By ruthlessly eliminating extra features, we have also given ourselves less to maintain and less to support. The focus on reduction of features has become known in software as YAGNI: 'You ain't gonna need it'. If we force ourselves to continually return to the hard facts of 60% of features never or rarely being used, we can keep driving down what we put into our products and focus on shipping the MVP as soon as possible.

Not all software is about features, of course. A great deal of our work in IT depends upon architectural choices. This tends to set up a paradox – do we take on the waste of over-investing in our architecture when it may turn out to be more than customers want, or do we worry more about the waste of relearning? Balancing these often depends on the type of market/arena you operate in. As Smith writes in his book Flexible Product Development, 'When change is rampant, YAGNI embodies considerable wisdom, when relative stability prevails, ignoring the future is foolhardy, especially when high cost of change items (such as architecture choices), are at stake.'

Activity 10: Mapping usage on your system

This activity requires working with colleagues or following usage patterns on your system. It should take place over 1-4 weeks.

Begin by selecting at least 10 features from your system. These could be anything from reporting features to recovery or version control.

Now try to plot usage. You can do this in a very low-fi way – chatting to colleagues and asking which they use daily, weekly or monthly – or you can track actual usage of those identified features via the system, including number of users, frequency of use and length of time using.

When you have the data, plot it out on a graph by frequency of usage.

You may want to present your results to the team. Are there any surprises? It is not unusual to find a few very low frequency items with high value (disaster recovery is a classic example), but on the whole most companies find at least one anomaly – a feature they have invested in heavily or which they think is important, but actual usage turns out to be low.

Are there any features which could be cut altogether? If there are, then cut them. This saves on maintaining complexity – another source of waste.

3.8. Relearning and hand-offs

In the olden days of advertising – before even Don Draper graced the streets of Manhattan – the art department was separate from the copy department. The copywriter would create some text about how marvellous the product was, write a headline – "Mum" is the word – and hand it down to the art department. Here someone would draw a picture and hand it over to layout who put the words and picture together and took it to the printer.

Figure 19. "Mum" deodorant ad from the 1920s

The whole process often involved huge amounts of rework when the original writer saw the picture and complained that he had a completely different kind of girl in mind, or that the layout split the text in the wrong place. Even if each person patiently explained their ideas and what they wanted to the next person in the chain, all the explanation took a lot of time and ideas were still miscommunicated.

In fact, the idea of separation between these departments became so absurd that in 1950 Bill Bernbach decided to put artists and copywriters together in a 'creative team'. Agencies hired 'teams', not two individuals, and both partners received awards together – or got fired for poor performance together! It was a concept that revolutionised advertising and working practices because the collaborative partnership achieved better creative results and significantly reduced waste.

In product development, Agile uses cross-functional teams to try and reduce hand-offs between different 'functions'. While individuals still specialise in particular areas – front-end development, back-end development, architecture, testing, etc. – everyone shares responsibility for the output. By working alongside one another, informal communication takes the place of formal hand-offs, meaning that most members of the team have a fairly good idea of what everyone else is working on.

Work still often needs to pass between teams, or requires expertise or input from specific individuals outside the team. The work is more than simply a product, it could be said to include all the tacit knowledge – the history and the learning that went into making the product.

Passing over the learning that goes alongside the work itself is what Poppendieck termed a 'hand-off'. If we do these hand-offs badly and fail to create organisational learning we frequently find ourselves coming up against the same problem and having to rediscover a solution. This is relearning. The more times we have to pour time and energy into 'reinventing the wheel', the more waste we are permitting. This happens not only at the macro project level, but also at the micro level between team members. How long does it take a programmer to get a tester up to speed, or a business analyst to 'translate' customer needs to the developers?

As Pavel Brodzinski writes, any hand-off 'means making [the] knowledge chain longer. It means adding another link between client/user and people who actually build the thing.' Each link in such a chain is a chance for delays, incorrect assumptions and miscommunication to creep in. Perhaps most importantly, it encourages a lessening of the attention to quality that is felt when you know that you, and you alone, are responsible for the software that is delivered to the customer.

In an effort to reduce the need for relearning, companies have invested time in documentation, processes and other paperwork.

Documentation as waste

THE AGILE MANIFESTO

We value...
Working software over comprehensive documentation

Figure 20. One of the values of the Agile Manifesto

Agile is often criticised for this principle. Working software is all very well, but companies also need to trace progression, check approvals, ensure learning can be passed on from one person to another, understand what the code is intended to do, how it has changed, and the thinking that went into developing it...

There is an important point here. When we develop a product we learn a great deal in every iteration. How do we ensure this learning is not lost when people move off the team or when one piece of work is moved from one team to another?

It's a really good question, but the way we most frequently answer it – with documentation – is flawed.

Companies spend an astonishing amount of time on paperwork.

Figure 21. Time to review the project initiation documentation!

Studies conducted by Capers Jones in the 1970s and 1980s reported system development costs split out by work category. One of the categories was 'paperwork', and it was deliberately split out from the thinking time required to put information on paper. Thinking activity was categorised separately as analysis, design, or test planning. So what did paperwork represent? Bureaucracy: activity with no value that can be assigned to it.

You may not be surprised to learn that this 'paperwork' turned out to be the second largest category for systems development work, accounting for more than 30% of the cost of production.

Mary Poppendieck attacks paperwork at some length: 'Paperwork consumes resources. Paperwork slows down response time. Paperwork hides quality problems. Paperwork gets lost. Paperwork degrades and becomes obsolete. Paperwork that no one cares to read adds no value… Just because paperwork is a required deliverable does not mean that it adds value.'

This is not to say that offices should be completely paperwork free. Some documentation is necessary – but you need to examine what it is being used for. The board may need something to make a decision on funding a project, but does that 'something' really need to be a PowerPoint deck of 400 slides? Who will actually use this? And is a presentation the best way of conveying information or assisting a decision? A lengthy file intended to capture all the learning from a project is likely to end up unread, gathering dust on a shelf. People capture knowledge more effectively than documents. Hand-offs inevitably involve the deterioration of knowledge. No matter how hard someone tries to explain how they do a job, they cannot transmit the 'gut feel' that comes from personal experience or personal relationships that make up a big part of the way human beings work.

When Pliny the Elder wrote his Natural History in the first century AD, he planned to transmit the whole of human knowledge for posterity. In some ways he did pretty well – we still have copies of his book today – which is how we know that the Romans had sophisticated pumping systems in order to mine gold at depth. But we do not know how those pumps worked – Pliny's explanations don't make sense to modern engineers. While in the middle ages, despite the fact that Pliny's work was widely read, mines regressed to the open-cast method. Nowadays, modern archaeologists digging at Roman mines have managed to match practical discoveries with Pliny's book to come up with a working model.

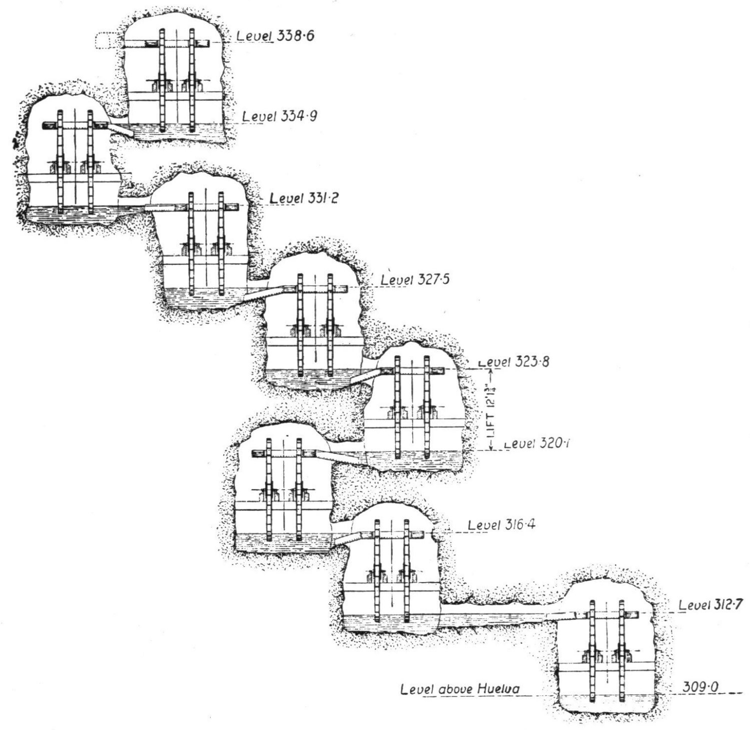


Figure 22. The sequence of wheels found at the Rio Tinto mines confirmed the descriptions in Pliny's work

What was missing was the 'tacit' knowledge that came from experience. Pliny's written description, however carefully researched and accurate, was just the tip of the iceberg. Exactly the same issue holds true for transmitting learning in any development process – documentation, reviews and process descriptions printed out or stored on a database, are unable to transmit the real learning.

Transmitting real learning

As with many solutions based on Lean principles, we really want to avoid creating the problem in the first place. Let's ask, why do we need documentation in the first place?

When documentation is required to explain how the code works, the likelihood is that the code was written badly in the first place.

> "A 1984 study measured the amount of time that we spend understanding code that we're tasked with maintaining at 50%. Poorly written code obscures its intent and behavior and forces us to relearn every time that we need to maintain it."
>
> **Matt Stine,** The Seven Wastes of Software Development

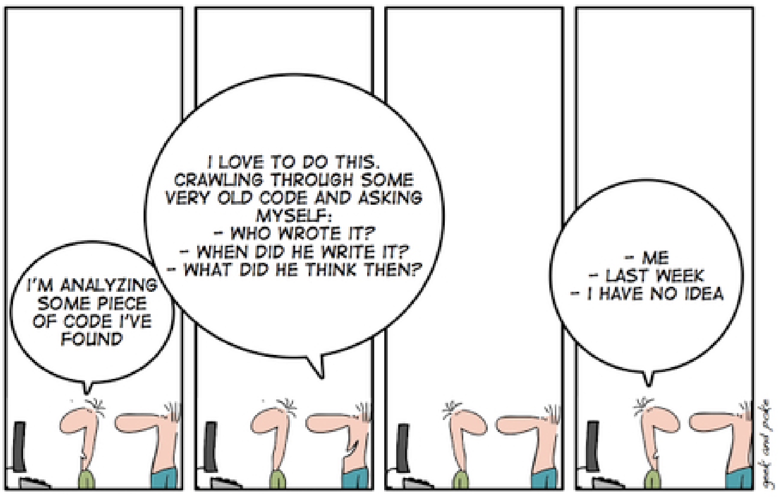

I'M ANALYZING SOME PIECE OF CODE I'VE FOUND
I LOVE TO DO THIS. CRAWLING THROUGH SOME VERY OLD CODE AND ASKING MYSELF: - WHO WROTE IT? - WHEN DID HE WRITE IT? - WHAT DID HE THINK THEN?
- ME - LAST WEEK - I HAVE NO IDEA
geek and poke

Figure 23. One year in an IT project - Day 8
Software Archeology (by Geek and Poke)

Good code, by contrast, is self-explanatory. Poppendieck writes, 'In software development, the tests and code are often just the right combination of rigor and conciseness to document the knowledge embedded in the software.'

What about other types of documentation, like progress reports, presentations and reviews? Often these are required because we have separated work into processes or phases, meaning testers are unaware of development work or senior managers have little involvement with the project.

Agile practices are designed to avoid much unnecessary paperwork. The daily stand up, task board, burndown chart and end of iteration review, for example, mean that every individual has a good understanding of the team's work and progress. As Poppendieck advises: 'If you must produce paperwork that adds little customer value, there are three rules to remember: Keep it short. Keep it high level. Do it off line.'

Where transfer of knowledge is required – whether between individuals on a piece of work or because someone is leaving the team – then rather than some kind of hand-off file or process, a more effective way of transferring learning is to allow people to pair together. Some companies might find this painful – paying two people for the same role for a month, for example. But the alternative – the loss of tacit knowledge – might be more costly in the long run.

Activity 11: The A3 report

This report got its name from the idea that an in-depth piece of work that resulted in real learning should be able to fit onto a piece of A3 paper. No more huge reports or files – just a sheet of paper to stick up on the wall.

Try this activity with a small team of 2-3 other people.

Select a problem that is causing the team difficulty and then work through the following steps using the template below.

BACKGROUND	TARGET
CURRENT SITUATION	ACTION PLAN
ANALYSIS	FOLLOW UP

Here are some questions that might help in each stage:

Background: Why did you select this problem? How big a problem is it? What other impacts are there on people, teams or processes?

Current situation: What have you observed about the problem in the existing work flow? Try and draw out the process as it exists as a diagram and symbolise problems as 'storm clouds'. Can you quantify the problem explicitly and measure its effects using specific metrics?

Analysis: What is the root cause? Try asking the 'five whys' (see the technique library) to make sure you get to the root of the problem.

Target: What do you want to see change? What is the outcome you desire? What counter-measures will help address the root cause to move you towards the goal? Identify the changes – draw out the new process as a diagram.

Action plan: Who will take responsibility for the implementation of each action? What sort of changes/outcomes do you expect to see? By when can you expect to notice a difference and thus update where you are?

Follow up: How will you check if your actions have made a difference? How will you know if you are moving in the right direction? Record actual results! What will your next steps be?

When you have completed the A3 report, put it up on the wall and share with others. As you take the actions as part of the implementation plan, be sure to record results and ask what people notice and how they feel about the changes.

3.9. Task-switching

Task-switching increases the number of hand-offs we have. Perhaps the company switches developers from one project to another in order to maximise capacity utilisation. The company thinks it is being efficient, in fact, the cost of task-switching and the additional hand-offs make for a great deal of waste (quite apart from the delays created by running at high capacity utilisation with lots of partially completed work).

We risk this ourselves when we switch from one task to another. You have probably had the experience of being in the middle of working on something when an email flashes up. You stop what you're doing to check the email and then can't remember how you were going to finish the sentence you were half-way through writing. Gloria Mark, Professor of Informatics at the University of California, found about 82% of all interrupted work is resumed on the same day, but it takes an average of 23 minutes and 15 seconds to get back to the task. Allowing interruptions, whether of email, phone calls, coffee with a colleague or another project – carries a heavy penalty for our productivity.

This doesn't just mean interrupts – the same cost is noticeable if we try to work across several different projects at once. If you need to get back up to speed on highly technical or complex knowledge, the switch may take up a huge proportion of your working day.

We don't subscribe to the idea that developers should have secretaries to answer their phones and bodyguards to stop senior management from wandering over to ask a question. Interruptions can be beneficial. If you are thinking hard about how to fix a bug and someone mentions an idea relating to the topic it can be helpful, even provide a creative spark. If a colleague asks if you'd like a sandwich brought up or asks if you've filled in your expenses form, this is unlikely to derail you completely.

Figure 24. Some developers wear hats signalling they should not be disturbed

On the other hand, developers (all knowledge workers actually) require an element of personal choice in dealing with the costs of task-switching. Being given the freedom to refuse interruptions and the focus to work on only one project at a time, allows developers to eliminate large quantities of waste.

3.10. Defects

In traditional manufacturing, a quality inspector would randomly inspect each palette of goods. Any defects meant that the whole palette must be checked product by product and then sent back for rework. To the Toyota Production System, this was the epitome of waste. How much more sensible to achieve zero defects in the first place!

Bugs in software are the equivalent issue. Finding defects in software, from bugs to a break in the build, suggests we are failing to build quality into the system appropriately. If we rely on a quality engineer or a test to catch our defects, we are handing off responsibility to the quality inspector – creating rework and waste. W. Edwards Deming was an early pioneer of design improvement. He famously joked that the way we use tests in order to remove defects is like someone saying, 'Let's make toast American style. You burn, I'll scrape!'

Instead, a Lean approach to software asks us to focus on ensuring the problem doesn't occur in the first place: rather than fixing bugs, we try to write error-free code; rather than spending lots of time maintaining legacy systems, we work to replace or refactor code as we go. In software we build in lots of feedback loops that explicitly attempt to find defects early on so that we can solve them quickly. Read the Discovering Quality session quality session for more details.

Finally, there is another kind of defect that leads to waste: 'failure demand'. John Seddon first identified this in his book I Want You To Cheat. He noted that in the 1980s when banks moved customer telephone calls from the local branch to large 'call centres', the number of contacts exploded. Banks had made the change in order to save money – by centralising the service they could be more efficient in both time and cost, but the far higher than expected volume of calls challenged this thinking. The reason for the larger volume of calls was that customers who had previously had their complaints or requests fixed the first time were now dissatisfied and needed to call back again.

Seddon argued that the higher volumes should be seen for what they were – evidence of failure demand, not valuable customer demand. He estimated that as high as 80% of all calls were due to this.

Many organisations see a rise in calls to the helpdesk after a software update or when a new interface is launched. Every call is a kind of waste. Worse, this failure demand is only the tip of the iceberg – for every customer who calls to complain, there will be far more who simply stop buying or using the service – arguably the most dangerous waste of all.

3.11. Helping you eliminate waste

Without wishing to sound like a waste awareness self-help group, the first step in eliminating waste is to know what it looks like and where to look for it. The examples above are intended to assist you in looking at your process and flow with a new eagle eye.

Dealing with waste often requires investment. This can be a difficult challenge for some organisations who are so focused on a narrow definition of efficiency that they fail to see the big picture.

When you were young, you probably drove an old banger of a car – belching fumes, always breaking down and running at just a few miles to the gallon. Now that you are highly respected in your profession, we'd like to imagine you have a beautifully fuel-efficient and reliable car driven straight out of the showroom. The difference is that when you were young you couldn't afford to buy a new car, the few hundred pounds that the old banger cost was more important than the fact that its running costs were three times as much.

Eliminating waste means thinking about investing to reduce it, and that means recognising that the types of waste we have mentioned are not one-off costs. They are recurring costs, like a car's running costs. The waste is a by-product of a problem in the system and it will continue to occur until we can fix the system.

In the Toyota Production System, every factory hand had permission to halt the entire assembly line in order to find out what had led to a specific defect and fix it. Try and imagine just how revolutionary this was at the time. A survey of 101 car manufacturers puts the average cost of stoppage at $22,000 per minute. This is close to a crime in most factories. By actively encouraging workers to stop the line in pursuit of zero defects, Toyota was making a really impressive commitment to quality and defect reduction. And this was not for some kind of PR opportunity, but because the company subscribed to the view that defects cost them more in the long run than stoppages.

Support, tools and training

We talked about the importance of focusing on valuable activities. This means your people must focus on valuable activities as well. It sounds obvious, but how much time do your developers spend filling in timesheets, updating progress reports, etc. when they should be creating code? We've talked about removing unnecessary activities, but what about when the activities are important but not valuable?

The first step tends to be to ask how you can cut down or remove non-value adding tasks. When such tasks are really necessary, the team can look at automating them with a variety of tools. Finally it may be worthwhile providing the team with some support to smooth away obstacles or carry out administration.

By investing in the right systems and software, we can save our people money. If you were a graphic design agency, you would accept that your designers needed high quality colour printers and that their screens needed to be specially calibrated. You wouldn't ask them to turn out award-winning posters using a 10 year-old monitor that flickered constantly, would you?

Unfortunately, such thinking is not uncommon. We've known organisations that give top-of-the-range laptops to the CEO and C-suite, but keep their IT department on outdated models. Investing in software that helps automate the build or deployment may not seem an obvious priority to a finance director who sees the license cost, but fails to appreciate the extra productivity of freeing developers up to do more valuable, creative work.

> 'Improving productivity is to invest in training. Experienced employees are more productive than new employees because they have come down the learning curve. Such learning curves can be accelerated by systematically transferring knowledge from the experienced employees to the new ones.'
>
> **Don Reinertsen**

To some extent training happens naturally and informally between colleagues, but there is also a more formal type of training which involves identifying skill gaps in the team and offering the development needed for people to acquire those skills.

Companies can reap significant benefits if they have generalists on the team. Reinertsen describes such people as T-shaped resources – expert in one function, but knowledgeable enough to help out in several other functions. Since such people are rare and therefore difficult to recruit, one of the best ways to gather a team of such people together is to train them. This, as Reinertsen points out, is something many organisations are reluctant to do, too focused on immediate productivity issues to appreciate the extra flexibility in capacity such a team would provide.

Activity 12: The ladder of investment

This activity requires the whole team and is done as part of your normal work.

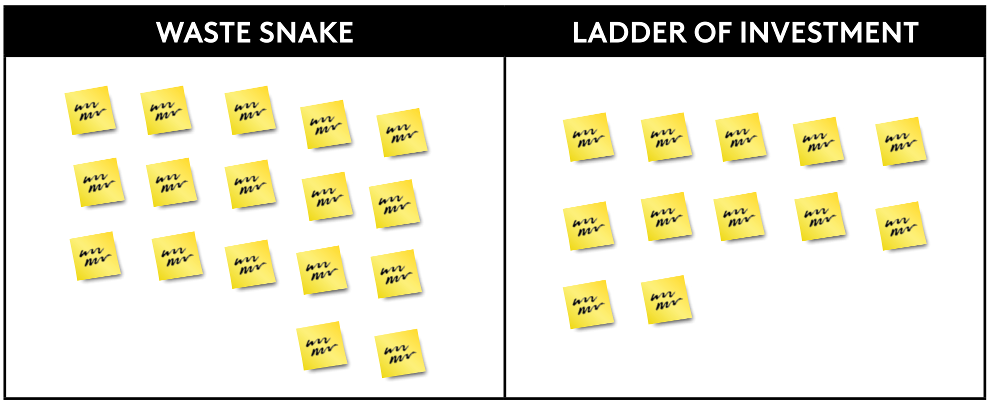
After you have been tracking the waste snake for a while, you may find that you incorporate various activities into your process which are directly about reducing or removing waste.

It can be helpful to track these and future ideas for waste removal as a 'ladder of investment', opposite to your waste snake. It forms a visual reminder of the work the team is doing to address and eliminate waste.

Add a 'ladder of investment' along side your waste snake and use it to track the things you want to improve over the next month.

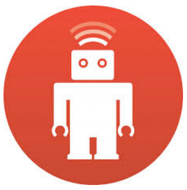
Here are some examples of things you might include:

- Break dependency on database in long running tests
- Request developer spec machines - standard build can't cope!
- Cross training for those that want it on RabbitMQ



We suggest you categorise the investment activities according to which type of waste they address or by more generic category 'training', 'refactoring', 'streamline release', etc.

CASE STUDY: Thoughtbot eliminates waste



Thoughtbot is an American software development house with an impressive roster of clients. They are dedicated to Lean Agile ideas regarding eliminating waste. Below is a selection of the initiatives they have introduced, both in-house and for customers, to reduce waste.

No more tracking of hours worked per feature

Thoughtbot decided that tracking how much time a designer spent on each feature or project was waste. As well as not providing any actual value to clients, designers and teams hated it as a piece of bureaucracy. This would be fairly radical for a great many development labs who either charge by the hour/day or who want to check that they're not spending too much time on a fixed price contract.

Reduce work on wireframes and other prototypes

Although the teams needed wireframes, etc. to help focus design and development, they also recognised that as artefacts they had no actual value. It was easy to overinvest in them. Thus they began to test using faster, more throwaway tools: paper prototypes, HTML mocks with Flutie, and sketching on an iPad. What was important was ensuring the team kept this 'design' phase to a minimum, learning what works cheaply on paper then iterating towards working software.

Justify every feature

Thoughtbot partners with customers helping them to test demand before building the feature. A good example is with TripAdvisor testing feature ideas by putting dead links on their home page and tracking how often they were clicked.

No waiting for management sign-offs

The team cut a major source of waste - the 'decision queue', waiting for sign-offs from product managers, marketing people, QA engineers, etc. No one was on the team except doers – designers and developers. Customers were asked to provide a single person responsible for quickly accepting (or rejecting) features. They keep a branch of apps in a deployable state so new features can go to production as soon as they are accepted. Thoughtbot even feels that this level of sign-off represented waste and that perhaps changes could be pushed out and then rolled back if they generated problems.

Minimise task-switching

Thoughtbot designers and developers usually work on one project per week. More than two projects each week is rare.

Eliminate defects that go undetected or unfixed

Thoughtbot keeps a suite of tools designed to ensure the team is alerted and can fix a defect quickly. The first step is to identify defects and notify the team (using a range of automated tools). Humans then determine the severity, and a team replicates the defect with a test, watch the test fail, write the fix, watch the test pass, and commit the code to version control. Since the code base has at least one branch of always-deployable code, the fix can go to production immediately if the defect is severe enough.

4 FLOW AT SCALE

In 1981, the well-known software pioneer, Barry Boehm wrote: ‘On a software project, there are an unlimited number of temptations to make the product, and the job, bigger and more grandiose. Avoid them. You’ll be much better off by reducing the scale’. In 2008, Larman and Vodde echoed Boehm’s sentiments in Scaling Lean & Agile Development, ‘After working for some years in the domains of large, multisite, and offshore development, we have distilled our experience and advice down to the following: Don’t do it.’

IT can make huge productivity gains through eliminating waste and optimising flow. At a point, though, IT projects also need to add capacity in order to get more done. The authors’ advice – well-meant and based on painful experience as it is – breaks down in the face of the really enormous challenges that we occasionally face. From massive enterprise systems for the UK’s National Health Service (NHS) or a state government to ambitious technological challenges like landing a man on the moon, there are times when a small team cannot deliver.

4.1. Lots of people and multiple teams lead to new waste

When you have many teams, the difficulty of coordinating their output is huge. The risk is that each team will forget about the larger value chain and instead optimise for itself (just as individuals sometimes do within the team). Teams can repeat one another’s work (leading to huge waste), or develop solutions that conflict with or hinder another team. In defiance of all common sense, adding more people can actually make a project slower. It certainly adds cost – and it also adds complexity. The worst case scenario is to have a series of teams all pulling in different directions, and thus achieving nothing.

Traditionally, the only answer to this was to have someone in control – a driver directing the many horses pulling the chariot, if you will. That meant a gradual concentration of power and control in those at the top who laid out what each team ought to do (meaning the work was aligned and – in theory at least – coordinated) and then gathering in and assembling the various ‘parts’ into a whole.

We all know, of course, that this traditional model often failed in practice. Trying to decide what everyone should do in advance led to massive up-front design, creating huge risk. Coordination problems continued, in any case. The creativity of individuals was never given free rein and decisions ended up stuck at bottlenecks, making projects almost unbearably slow and inefficient.

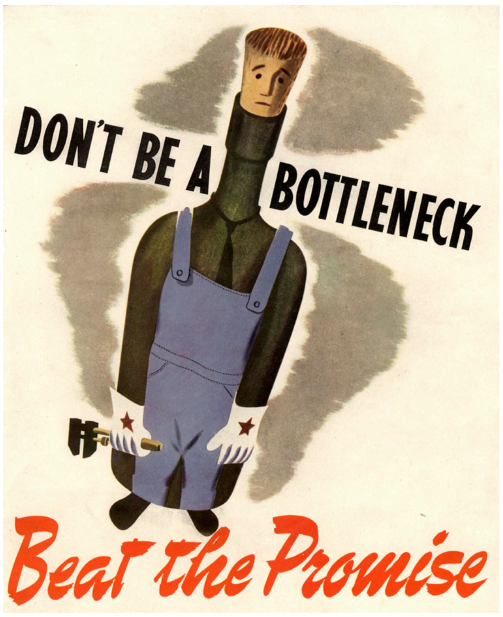


Figure 25. "Beat the promise" was the slogan of RCA's production incentive program during World War Two

Agile's rather glib answer was that every problem or project should be broken down into discrete, loosely-coupled elements that can be solved by individual teams with minimal collaboration between them, and only over-arching coordination. We say glib, because such a solution is far from simple to implement. Sometimes it is impossible: some problems are tightly coupled; some dependencies cannot be broken, and the effective coordination of many teams can become almost indistinguishable from a centralised 'command and control' style of management – with all its associated risks and costs.

Thus, we need to learn how to stay productive at scale. We need to structure our entire organisation as well as our projects in a way that supports a disciplined approach to flow and to the communication, collaboration and coordination that enables flow. This should be a structure that focuses on the framework within which decisions can be made, while leaving teams and individuals free within that to decide what to do. The aim is to capture the improved speed and collaboration within small, discrete teams, without sacrificing the possibilities of productivity offered by massively increased capacity at scale.

4.2. A formal structure versus the decision framework

The Scaled Agile Framework (SAFe) offers a one-page visual breakdown of how the differing levels of a hierarchy sit together to offer nested goals, nested teams, multi-level plan and work breakdown. In essence, it provides a 'hybrid' style of management between autonomous teams and a structured hierarchy.

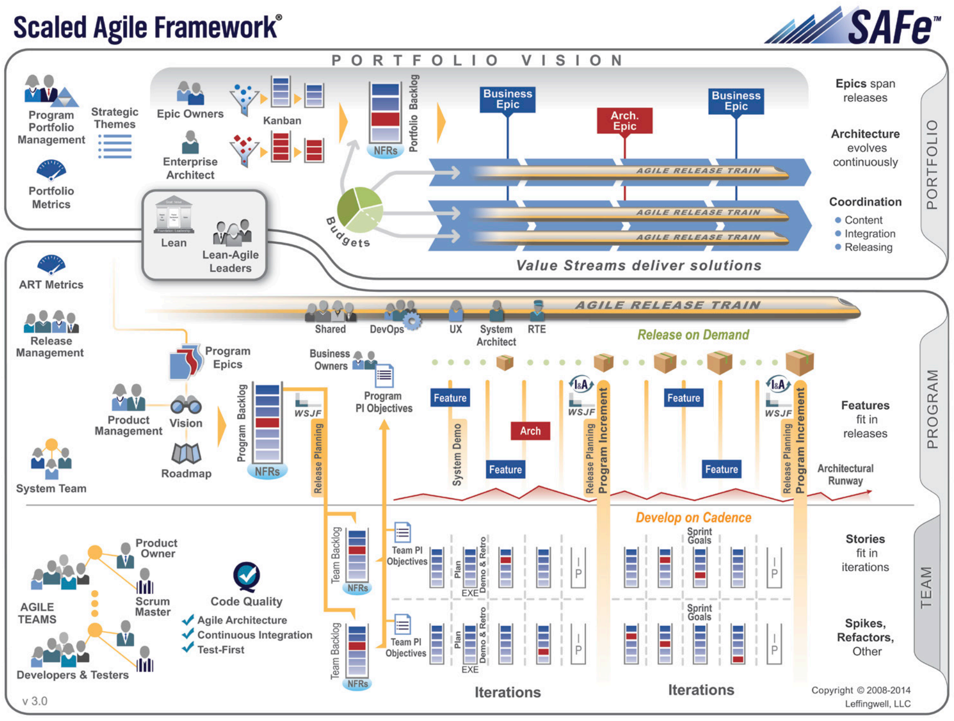


Figure 26. Management between autonomous teams and a structure hierarchy
(Copyright © Leffingwell, LLC and Scaled Agile Framework™)

The organisation is sliced at a portfolio, program and team level, with the goals and objectives in each aligned. The portfolio backlog items are characterised by business epics – large developmental ideas or slices of customer functionality which will span releases. These are decomposed into a program backlog (split by features), which in turn flow into team backlogs, normally formed around feature teams rather than component teams. Just as a product idea is split into increments and then iterations, the ideas is that the organisation as a whole can be set up to reflect the same division.

It's worth noting that there is nothing revolutionary about this model. It describes, in essence, a standard hierarchy overlaid with a Scrum or Agile working method.

The key trick continues to be whether the 'business epic' can be split down into separate features and whether the organisation can harness the knowledge of those actually doing the work with the ability to make system-wide decisions that involves the work of multiple teams.

The Scaled Agile Framework relies on an 'Agile Release Train' to coordinate many of the outputs from feature teams, trying to optimise across all the development teams (not just for a single team). They create roles that act – essentially – as coordination specialists: suggesting a Release Train Engineer, Product Management, System Team, Release Management, UX Designer and System Architect, for example. While teams are responsible for their own planning, scope and schedule, they all work to a standard development and release cadence.

The Release Trains are linked back to value streams (delivering business value) which are, in turn, linked to the original portfolio vision. These have their own specialised coordination roles, collectively known as the Portfolio Program Management, who make decisions about how to allocate and assure funding, drive the programmes and supply due governance.

This structure has some major benefits. It is clear and it recognises the importance of making different types of decisions at different levels. Decisions are best made where the information is concentrated. In software development, engineering and technical decisions are thus often best made by the developers, while major financial decisions about what to invest and when to stop are often better made by those with the financial information.

ANY QUESTIONS ABOUT YOUR WORK?
...Ask your Supervisor
LABOR-MANAGEMENT COORDINATING COMMITTEE

Figure 27. Formal structures can come with a communication and collaboration cost

Thus, a framework such as SAFe, backed by the kinds of practices described within VFQ, can be helpful for many organisations. But it does not remove the risks of a traditional hierarchy – every 'coordination' layer offers more opportunities for wait time, dropped communication and other forms of waste. There are still numerous ways that flow can become blocked.

Don Reinertsen explores a more flexible concept. He suggests creating a framework that takes account of where types of information are concentrated. He describes a series of financial 'heuristics', set at a senior level, within which developers or teams have the framework within which they can make the correct decisions. If a team needed to decide whether to spend more money to go faster, for example, the organisation should have laid out in advance what might be acceptable expenditure given an agreed cost of delay. Only on very marginal decisions would a team need to seek further authorisation and therefore incur the resulting coordination cost.

CASE STUDY: Boeing structures the work to build the 777

When Boeing was designing their 777 plane, they had already given various performance guarantees to their main customer, United Airlines. In particular, Boeing had agreed to pay any excess fuel required for the plane's lifetime if the final plane came in over a certain weight/efficiency. About half way through the design phase, Boeing checked on the weight – and discovered they were in serious trouble. They instantly began looking for a way to reduce weight, although naturally while still designing a cost-effective plane to manufacture.

Now, as Don Reinertsen, who worked with Boeing, points out, the typical response would have been to expect the Program Director to make system-level decisions about where and how weight should be reduced. This would have taken up enormous time as he had to make decisions about details – in effect micro-managing his engineers.

Boeing had begun the project by trying to break down dependencies. The initial requirements had allocated each team (wings, tail, engines, etc.) a target weight and budget. The problem was that these allocations were not quite right – and of course, while dependencies did exist between the teams, the system had not been set up to promote collaboration. If an engineer on the wings realised he could make a weight saving for a fairly cheap design cost, he might not motivated to do so unless his team was approaching limits on the 'wing weight allocation' budget. Meanwhile, an engineer on the tail, already at their weight allocation budget, might be willing to spend huge amounts of money to save just a little bit of weight. It's a classic example of how a project can be structured to promote independence but fail in collaboration and coordination.

Fortunately, Boeing realised what was wrong and set up a different structure. All engineers needed to save weight. Any engineer could spend up to $300 to save each pound of weight. There was no need to get approval or waste any time even thinking about it – if the weight saving could be made, then the engineer should do it.

If the weight saving cost more, then the engineer should ask his supervisor (who had a slightly broader view). The supervisor could approve weight-saving up to a cost of $600 per pound. Beyond this, the Program Director's approval was required for up to $2,500 per pound. This was because these more expensive weight savings might well be justified by the knock on effect they would have for the rest of the design, but only those with an overall view of the design could make the decision.

As Don Reinertsen says, the beauty of Boeing's system was to structure the work and the project organisation so that the decision-logic had been agreed at a system-level. The organisation had excellent overall control, but the actual decision-making happened on the ground, by the independent engineers. Everyone was making decisions against a well-understood overall goal – to get weight down within an acceptable budget. The decision-logic ensured that 'acceptable' was understood and adhered to.

Activity 13: A decision framework

This activity will require working with several others within your organisation.

Take the cost of delay figure you calculated in Activity 6. Make sure that you have consulted widely to reach this figure and that you have recorded (and tested) as many of your assumptions as possible.

Now consider what decisions require some form of permission within your project. Do any of these ever cause delays or 'queues' within the process? Do you ever feel that decisions are made which optimise for a part rather than the whole?

If the answer to these questions is yes, then it is worth seeing if you can create a financial framework to enable speedier decision making by those doing the work.

What amount could the team spend, for example, in order to reduce a certain amount of time in development or to eliminate a source of waste? Find the people who would need to sign off on such a change and see if you can trial the idea. Monitor the results and then present back.

You will need to show that the decision framework did speed up flow and that it helped the team develop faster and provide more value. Be very aware that the success of your framework depends upon the accuracy of your numbers. This doesn't mean you should spend ages coming up with the figure in the first place, rather that you should be meticulous about constantly refining your numbers as you discover more.

5 CONCLUSION

We've seen how speed is an issue of increasing importance for many organisations. It's a common error to imagine that this is an entirely modern phenomenon – in fact the need to out-innovate and act faster than your competitors has always been part of business. Have you ever been to see the Cutty Sark docked in Greenwich? This was one of the fastest tea clippers, designed to get the first tea of the season from China to London – with a substantial bonus to the captain who got the first cargo ashore and into the teacups of the waiting aristocratic ladies of London.

Figure 28. The Cutty Sark docked at Greenwich

If speed has always been important, it has also always been dependent upon business outcomes. The Cutty Sark sailed fast to win the bonus, driving the sailors hard, losing spars, running the sails ragged and, on one occasion, even losing the rudder... the cost in repairs and wear and tear was huge! When the Suez canal opened, steam ships could do the journey in a shorter time and with a larger cargo. Tea had begun to lose its aristocratic image as more and more people drank it, and finally the less romantic sight of a steam ship meant people no longer bet on the outcome of the race... speed was no longer the point for the tea trade – it was all about throughput.

Whether you need to travel at extraordinary speed (the Ferrari of the development world), or whether you need to maximise your throughput (there's nothing wrong with being a bus), then optimising your flow will assist you. These choices are discussed in greater detail in the session book Trade-Offs.

Yet no matter what your desired outcome – even if you have a set deadline with no possible benefit for delivering early, then eliminating wasteful delays and optimising the whole will still make your planning more reliable and your overall process more effective.

Learning outcomes

Now that you have completed this session, you will be able to:

Appreciate why time is so important to an organisation and why the customer's view of time matters

- Delivering faster offers a competitive advantage
- Most organisations focus on trying to speed up delivery where time is already under pressure
- 'On time' means different things to different people and depends on how much it costs as well as many external factors
- Only the customer's time sensitivity really matters. Remember that speed is not an end in itself – the valuable outcome is what matters, sometimes delay is correct

Consider the benefits and limitations of the 'on time' delivery approach adopted by most companies

- 'On time' is rarely set by a genuine customer need, but is an internal concept
- Structures, contracts and metrics reinforce the internal view – these can have unintended consequences to quality and delivery
- To guarantee a delivery date, most organisations introduce buffers which are used on valueless activities, resulting in a longer development time

Analyse the cost of delay in development projects

- Recognise that delay is an opportunity cost to your organisation
- Know how and when to make calculations about the cost of delay to inform decision making

Take a broader view of end to end flow

- Optimising the whole is always more efficient than optimising any one part
- Consider the full value-chain, not only development, to make large gains in delivery speed
- Understand the link between cost of delay and capacity and the importance of not running at full capacity utilisation

Appreciate the opportunity offered by flow optimisation to eliminate delay and speed delivery

- Recognise the different types of waste to be found in software development
- Appreciate the need to operate with slack and the difference between this and waste
- Consider a range of tools to help eliminate each type of waste in software development

Judge how to optimise flow at scale

- Appreciate the difficulties in structuring the organisation to support large scale Agile projects across multiple teams
- Consider the benefits and potential costs of formal hierarchical structures
- Create decision frameworks to assist in optimal flow at scale

BIBLIOGRAPHY

Agile Manifesto, 2001. Principles behind the Agile Manifesto. [online] Available at: <http://agilemanifesto.org/principles.html>. [Accessed 22 March 2012].

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

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

Brodzinski, P., Hand-Offs Are Bad (But Unavoidable). [online] Available at <http://blog.brodzinski.com/2011/12/handoffs-are-bad.html>. [Accessed 7 March 2012].

Croak, D., 2011. Eliminate Waste. [online] Available at: <http://www.greenhornconnect.com/blog/dan-croak-agile-principles-practice-part-i-eliminate-waste>. [Accessed 17 September 2013].

DeMarco, T., 2001. Slack : Getting Past Burnout, Busywork, and the Myth of Total Efficiency. Dorset House Publishing.

DeMarco, T., Lister. T., 1999. Peopleware: Productive Projects and Teams. 2nd Edition. Dorset House Publishing.

Domingo, R., 2003. True Productivity: The Key to Profitability. [online] Available at: <http://www.rtdonline.com/BMA/MM/10.html>. [Accessed 15 December 2011].

Drucker, P., 2007. The Effective Executive. 2nd Edition. Butterworth-Heinemann.

DZONE, 2010. The Seven Wastes of Software Development. [online] Available at: <http://agile.dzone.com/articles/seven-wastes-software>. [Accessed 7 March 2012].

Fast Company, 2008. Worker, Interrupted: The Cost of Task Switching. [online] Available at <http://www.fastcompany.com/944128/worker-interrupted-cost-task-switching>. [Accessed 7th March 2012].

George, M., 2004. Conquering Complexity in Your Business. McGraw-Hill Professional.

Harvard Business Review, 1993. Japan's Dark Side of Time. [online] Available at: <http://hbr.org/1993/07/japans-dark-side-of-time/ar/1>. [Accessed 1 December 2011].

Israel, G., 2010. Standish Group Chaos Reports Revisited. [blog] Available at: <http://theagileexecutive.com/2010/01/11/standish-group-chaos-reports-revisited/>. [Accessed 11 January 2012].

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

Morris, W., 1974. Beauty of Life: An Address Delivered at the Town Hall, Birmingham, in 1880. Brentham Press.

New Product Dynamics, 2011. New Product Development And Fast Cycle Time. [online] Available at: <http://www.europa.com/~preston/>. [Accessed 28 November 2011].

Oxford Dictionaries, 2010. Oxford Dictionary Of English. 3rd Edition. OUP Oxford.

Pliny, G., 2004. Natural History: A Selection. Penguin - reprint edition.

Poppendieck, M., Poppendieck, T., 2004. Lean Software Development: An Agile Toolkit. Addison Wesley.

Poppendieck, M., Poppendieck, T., 2006. Implementing Lean Software Development: From Concept to Cash. Addison Wesley.

Poppendieck, M., Poppendieck, T., 2009. Leading Lean Software Development: Results Are Not The Point. Addison Wesley.

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

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

Reinertsen, D., 2012. Decentralizing Control: How Aligned Initiative Conquers Uncertainty. Presentation given at the Lean Software and Systems Conference 2012. [online] Available at: <http://vimeo.com/45947817>. [Accessed 19 September 2013].

Ries, E., 2011. The Lean Startup: How Constant Innovation Creates Radically Successful Businesses. Portfolio Penguin.

Scaled Agile Framework. The Scaled Agile Framework. [online] Available at: <http://www.scaledagileframework.com>. [Accessed 24 September 2013].

Seddon, J., 1992. I Want You to Cheat!: The Unreasonable Guide to Service and Quality in Organisations. Vanguard Consulting Ltd.

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

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

Smith, P., Reinertsen, D., 1997. Developing Products In Half The Time: New Rules, New Tools. John Wiley & Sons.

Stalk, G. (Jr.), Hout, T., 1990. Competing Against Time: How Time-Based Competition Is Reshaping Global Markets. Free Press.

The Standish Group, 1995. The Chaos Report. [online] Available <http://www.projectsmart.co.uk/docs/chaos-report.pdf>. [Accessed 28 November 2011].

Wikipedia, 2011. Declaration Of Interdependence. [online] Available at: <http://en.wikipedia.org/wiki/PM_Declaration_of_Interdependence>. [Accessed 1 December 2011].

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

VFQ Value Flow Quality

valueflowquality.com

emergn.com

by emergn