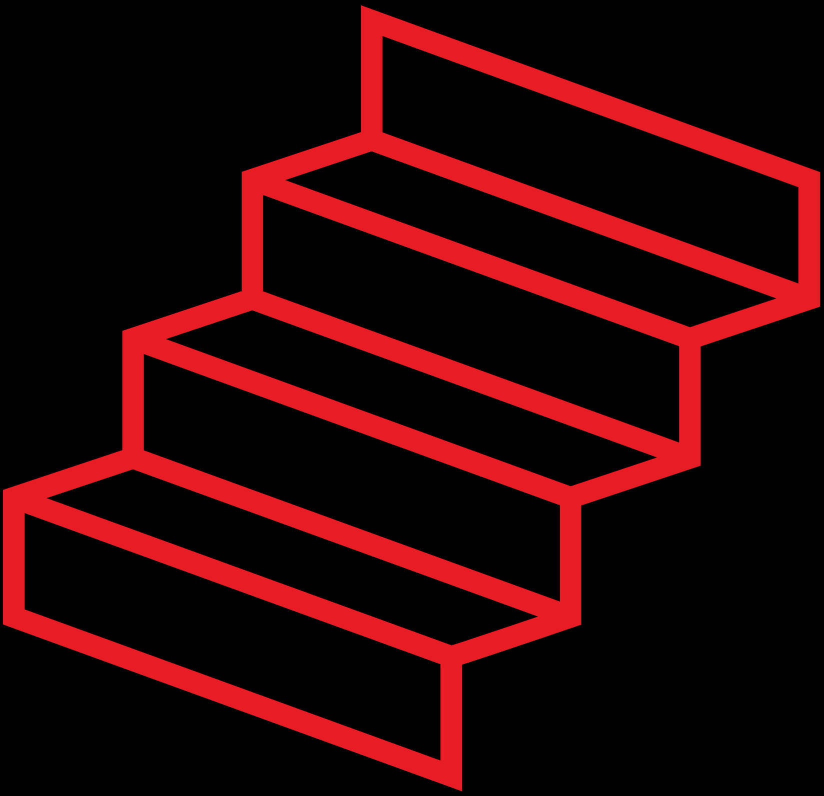
VFQ Value Flow Quality



Delivering early and often

by emergn

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

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

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

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

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

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

CONTENTS

INTRODUCTION

In the Prioritisation session we talked about the importance to an organisation of delivering value early: not because it's a nice thing to do, but because it enables you to make more money sooner. In particular, we showed how the financials of a business case can radically change when you begin delivering a piece of value early. In this session we will focus on the method of delivering that value, primarily achieved through slicing an idea into increments.

This can be hard for people to get their heads around. Oh everyone understands why it would be useful – who wouldn't want to start selling something early and improve cash flow by gaining revenue sooner? But we are very used to the idea of products being delivered whole. It takes an imaginative leap to see how these might be broken down and delivered differently.

Many managers have signed on for the idea of incremental delivery without really understanding what they might have to do. After being asked to think about which element they could deliver early, they quickly start explaining why incremental delivery won't work for their particular project. This project is just so complicated and filled with dependencies that it can't possibly be split into increments. They look around for examples to prove the point. How about a car? they say triumphantly. You can't deliver that in increments, can you? Wheels without an engine are useless; a car chassis without a body is pointless!

Figure 1. Cars in production, not ready for delivery!

This is how we're used to thinking about products. They seem indivisible to us. First of all, of course, software is not a car. Bits on a hard-drive can be configured more easily than car parts. Yet, even a car can be imagined differently if we try hard enough. Design enhancements to an original model can be delivered over time – brakes that last longer, a higher performance engine, an optional sunroof – together these changes can add up to a bigger change than launching a whole new model. Or we could look at it another way – the car parts do have independent value and could be sold to other manufacturers or as 'kit cars'. Or we could launch our car to the UK market only, and not worry about fitting left hand drive for another year. In short, there are many ways of considering incremental delivery – even with something as physical and difficult to divide as a car.

The exercise is far easier with software. Most elements of software can be delivered separately and still have value. A complex software application has value before it is complete: for example, a piece of search functionality might begin with a simple search for a name, but it might not yet allow us to do anything with the resulting list, or even order it. Yet the list of people called 'John Smith' in our total database might well be of use to us. A second characteristic that makes software perfect for both incremental and iterative development, is that it is inherently malleable. Unlike the metal and plastic of car parts, which once moulded into a shape are very hard to reconstruct into a different design or size, software can expand and contract, be reconfigured or changed or updated.

By the end of this session you will:

1. Understand how increments and iterations are defined.
2. Appreciate the benefits of delivering early and often:
3. financial, marketing, engineering and other.
4. Identify examples of where incremental delivery is used as a successful business model.
5. Begin to break projects into increments through:
6. developing a mindset to split dependencies and ideas
7. using different prisms to split ideas: value, risk, stakeholder, urgency, geography, necessity.
8. Know how small to go with your increment.
9. Be able to map a customer journey or story strand to create slices or bites of functionality.
10. Put in place practices that enable delivering early and often.
11. Balance potential drawbacks or limits to delivering early and often including transaction cost and technological break-through.
12. Demonstrate the value of incremental delivery.

1 SO WHAT IS DELIVERING EARLY AND OFTEN ANYWAY?

Delivering early and often describes the development style known as Incremental and Iterative Delivery. The Agile Alliance's Guide to Agile Practices defines an incremental development strategy as 'each successive version of the product is usable, and each builds upon the previous version by adding user-visible functionality.' The 'version' is the increment: usable software that has an innate value. The Agile Manifesto states, 'Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.' And all Agile methodologies stress the importance of an incremental approach to software delivery – that is, to delivering value and delivering it early.

Incremental delivery is such an important element of Agile methodologies that there are several different terms which you may hear – all of which essentially mean 'increment'. In Scrum, the increment is called 'potentially shippable product'. Other methodologies use alternative terms. You may also hear people talking about 'Minimum Marketable Feature' (MMF) or 'Minimum Viable Product' (MVP). The terms draw attention both to size (small) and to its independent value (marketable or shippable – that is, something a customer can use).

It's importance to Agile doesn't mean that more traditional waterfall methods can't also do incremental delivery of value. Many have a phased launch delivery plan, which is essentially a form of incremental delivery. Because the phases tend to be made up of large batches, the value is not being delivered as early as it could be – the increments could be smaller. We will go on to discuss the problems this can cause in a further session – Batch Size Matters.

Delivering an increment requires you to look at the structure of a project in a different way and this can often require an imaginative leap. Instead of the traditional horizontal process (based on supposed efficiencies of large batches), incremental delivery asks us to slice our product vertically and deliver sections of value.

Activity 1: Increments in your organisation

This activity will take 10 minutes. You can do this activity on your own or as part of a conversation with colleagues.

Consider a current project, was it delivered in increments? If so, consider of what the first increment consisted (think about the features that were released to customers, for example).

Reflect on how the project could have been split differently and what the effects would have been. Could you have launched faster? What might you have learnt sooner?

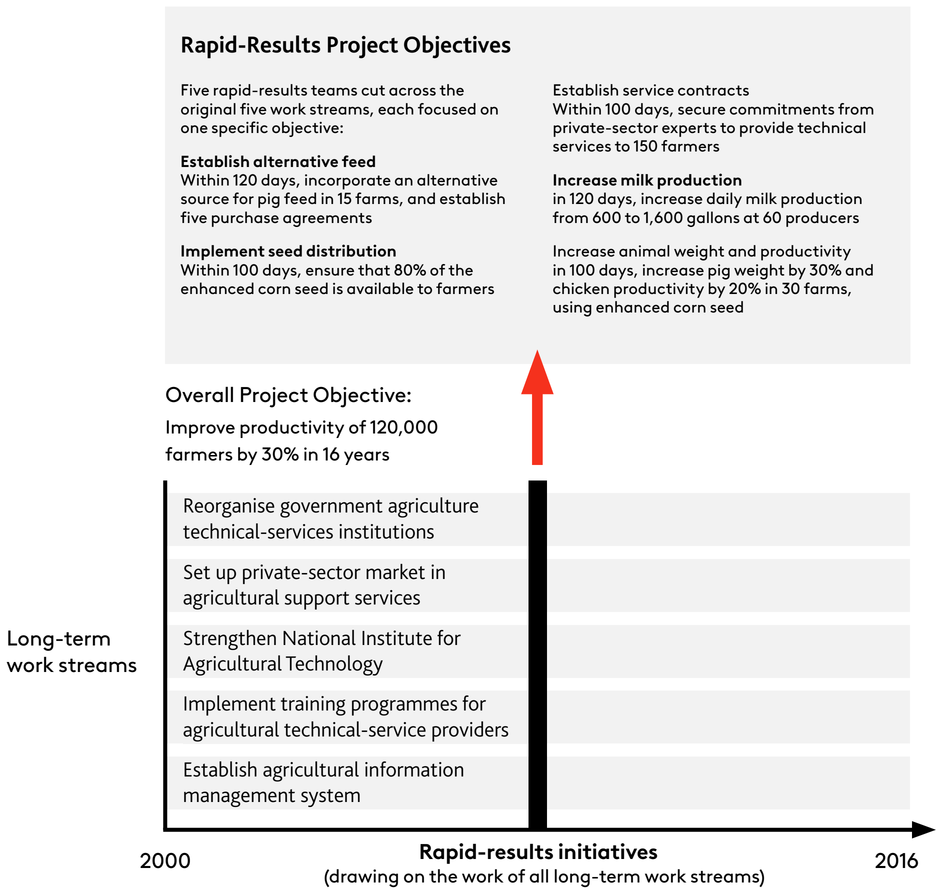
CASE STUDY: World Bank

The World Bank was considering a project to improve the productivity of 120,000 small-scale farmers in Nicaragua by 30% in 16 years. A team of World Bank experts and Nicaraguan Ministry of Agriculture officials began conducting surveys and analysing data. They were creating a long-term plan that would help huge numbers of farmers meet the goal. There were various long-term initiatives the planners believed would stimulate productivity. Implementation could then begin. This is what we would describe as horizontal slicing of the project – analyse, plan, implement, observe results.

It was taking a very long time. So long, that the World Bank also sliced the project vertically. They created little mini-projects to deliver small increments of value as an immediate pay-off and also as test-case pilots from which they could learn for the longer-term plan. After all, they realised, success in raising production for just one farmer in year 1 of the project is as valuable as raising productivity for 16 farmers by year 16. There would also be a positive effect on the project's reputation and morale, increasing the likelihood that other farmers would sign up to the plan.

One such increment was to increase Grade A milk production in the Leon municipality from 600 to 1,600 gallons per day in 120 days with 60 small and medium-size producers. The team with this goal discovered early on that the issue was not production, but quality. The farmers could produce the quantity of milk, but distributors had to throw away nearly half due to contamination, spoilage and other problems. The team brought in a representative of Nicaragua's largest private company in the dairy sector – a potential customer, in other words, who could state what quality standards were necessary and thus what hygiene standards and testing equipment needed to be introduced for the farmers. Improvements were made rapidly, but a new 'quality' issue raised its head. Small farmers didn't have the storage facilities to keep the extra 'Grade A' milk now being produced, while investing in refrigeration facilities would be an excessive capital investment. The customer representative on the team took the opportunity to change from being an advisory potential customer to a real customer. Now his company could be assured of a quality supply, he was prepared to have the milk collected daily rather than twice weekly.

The overall goal may well benefit from more farmers adopting the same practices and the project can clearly learn from these results, but the increment's value does not depend on these further benefits. The increase of 1,000 gallons of milk a day has independent value in and of itself.



Rapid-Results Project Objectives

Five rapid-results teams cut across the original five work streams, each focused on one specific objective:

Establish alternative feed
Within 120 days, incorporate an alternative source for pig feed in 15 farms, and establish five purchase agreements

Implement seed distribution
Within 100 days, ensure that 80% of the enhanced corn seed is available to farmers

Establish service contracts
Within 100 days, secure commitments from private-sector experts to provide technical services to 150 farmers

Increase milk production
in 120 days, increase daily milk production from 600 to 1,600 gallons at 60 producers

Increase animal weight and productivity
in 100 days, increase pig weight by 30% and chicken productivity by 20% in 30 farms, using enhanced corn seed

Overall Project Objective:
Improve productivity of 120,000 farmers by 30% in 16 years
Long-term work streams
Reorganise government agriculture technical-services institutions
Set up private-sector market in agricultural support services
Strengthen National Institute for Agricultural Technology
Implement training programmes for agricultural technical-service providers
Establish agricultural information management system
2000
Rapid-results initiatives
(drawing on the work of all long-term work streams)
2016

Figure 2. The World Bank's project plan

1.1. The difference between increment and iterate

You say increment, I say iterate (let's call the whole thing off). These two terms are frequently bandied about as if they were identical, but in fact the two concepts are quite different. In essence, an iteration is a process designed to get you towards the unit of value – the increment.

Craig Larman in Agile & Iterative Development: A Manager's Guide writes, 'Incremental delivery is often confused with iterative development. A six-month delivery cycle could be composed of 10 short iterations. The results of each iteration are not delivered to the marketplace, but the results of an incremental delivery are.'

Each iteration is a complete mini run through the development process, incorporating analysis, design, programming and test. An iteration aims to produce an increment of software, but it may not necessarily do so. In fact, you might take a step back and say that an iteration really exists in order to elicit feedback. At the end of the iteration, we should have a piece of working software. Now we need to test/demonstrate this software with a customer. It is quite likely that we will need to modify the software.

'Oh, I see!' the customer may exclaim. 'I love how it looks but that's absolutely useless to me.' The high-quality piece of working software is not, in other words, a shippable increment. A new iteration is required to work further on the software, incorporating the customer feedback. (And on a quick pedantic note, iteration comes from the Latin, meaning to do again, to repeat. If you hear anyone saying that they need to 'reiterate' or even worse 'to reiterate over again' they are guilty of tautology and should probably be smeared in honey and buried in an ant nest.)

Jeff Patton uses the Mona Lisa to illustrate his point:

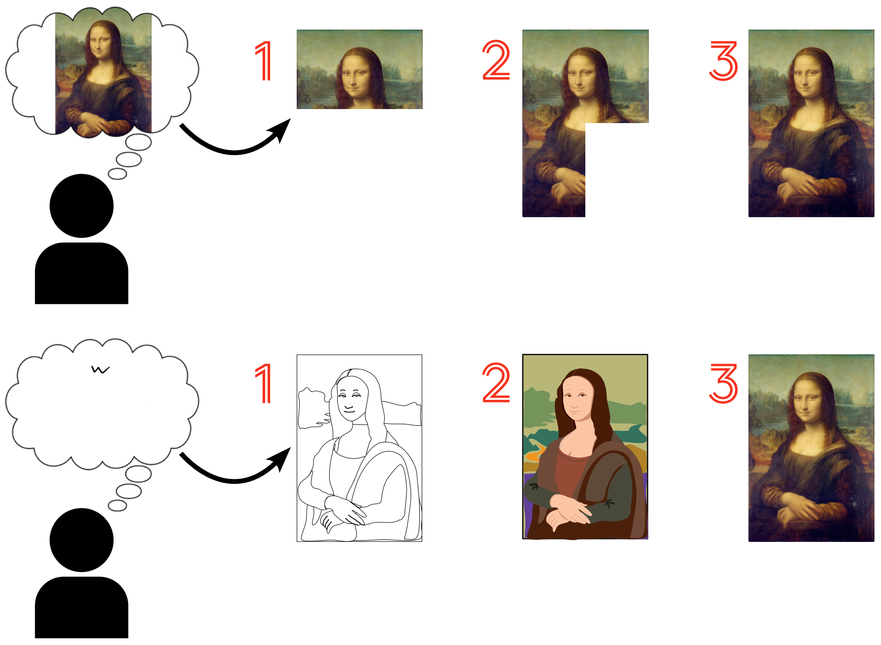


Figure 3. An incremental versus an iterative and incremental delivery approach

Artists work their way towards the final picture by a series of iterations including sketches, false starts and rough blocking. They don't start from a corner and work inch-by-inch without mistakes until they've covered the whole canvas. Software, like art or writing, is a creative process, and is especially suited to iterative development – a journey towards the finished product. Software development is also rather more costly than starving in a garret while you work on your masterpiece, and for this reason, software tries to deliver parts of value early. This is rather like the artist who managed to sell the sketches before the finished picture was ready, or the writer who sold the story in serialised chapters as he wrote it (as Dickens did for most of his novels).

Since iteration is all about gaining feedback in order to improve the solution, it follows that we want to iterate quickly. The focus is not on creating a complete and perfect design, but rather on an end-to-end piece of working software which we can then validate. On greenfield projects, this could be a walking skeleton – something which you have no intention of ever releasing externally, but which can provide significant internal value by validating the architecture. On brownfield projects, the iteration might be more about improving design than changing behaviour. It can be as simple as a clean-up or performance tuning – making the architecture more robust or scaling the ability of the software to cope with more users, for example.

Alaistair Cockburn sums up by pointing out that since incremental development works on a cycle, it helps you improve your process. But the choice of process depends on you. 'It neither implies, requires nor precludes iterative development or waterfall development – both of those are rework strategies. The alternative to incremental development is to develop the entire system with a 'big bang' integration.'

CASE STUDY: The appeal and risk of the big bang

boo.com

Boo.com is famous. Or perhaps infamous is a better term. During its glory days, this sports/fashion online retailer was named as one of 'Twelve Start Up Superstars' and was lauded in The Wall Street Journal, the Financial Times, Fortune magazine and Vogue, amongst many others. Now the company tends to be named on other lists, 'Boo and the hundred other dumbest moments in e-Business History', 'The top 10 dotcom flops' and even the founder's own book *Boohoo 'from concept to catastrophe'*.

For anyone who was asleep or at school during the dotcom boom and crash, boo.com was seen as embodying the excessive consumption and management errors of an era. They burned through $135 million of investment in 18 months, (to be fair, Ernst Malmsten, the CEO, denied charges that they wasted money on champagne, by pointing out he only drank vodka). There were usability troubles with the website and glitches with the technology platform – the 3D imaging and fancy graphics were unbearably slow at a time when most people had dial-up internet connections. Flaws in the business model, like the number of returns, undercut expectations. Boo.com had a vast staffing cost, employing 400 people in multiple countries around the world. Because it was launching in numerous countries with their own tax laws, languages, and fulfilment methods, the system set-up complexities were enormous. Dealing with these took time (delaying launch), and required vast technical support. A blog from former employee Tristan Louis, drew several lessons, including: 'Plan early, think of all that can go wrong, and then plan it again. Usually, spending more time on specs saves you from many headaches down the road.'

It's a tempting conclusion – perhaps the one that most employees would have reached. But actually, such a lesson is symptomatic of what was wrong at the heart of the business. The planning may not have been perfect, there were management errors and technical messes, but the business's original sin was to turn its back on incremental delivery. As Ernst Malmsten admitted, 'instead of focusing single-mindedly on just getting the website up and running, I had tried to implement an immensely complex and ambitious vision in its entirety. Our online magazine, the rollout of overseas offices, the development of new product lines to sell on our site – these were all things that could have waited until the site was in operation.'

These things could certainly have waited, but the website that Malmsten envisaged was still fronting an overambitious model. Incremental delivery would have highlighted problems earlier, from the user experience to returns. By the time boo.com collapsed, it was turning over $500,000 in orders every 2 weeks with good conversion rates and a healthy repeat purchase rate. These were great figures – but too late. A revenue stream that had come in earlier, even a smaller stream, would have offset the huge quantities of investment being poured in.

So why hadn't Malmsten considered incremental delivery? Did he just not think of it? Possibly. He certainly made plenty of mistakes. But we should also point out that the decision not to use incremental delivery was tempting. There were several reasons why Malmsten and his advisors were seduced by the big bang.

Suppliers – the people that boo.com needed to win over in order to have something to sell – were attracted by the idea of a global company. They weren't interested in dealing with an internet company who simply seemed like an alternative distribution channel for a single country. The investors also wanted to deal with a global company – time and again the feedback Malmsten received was that the global reach was attractive and exciting to large investors. The team were worried about competition. They wanted to raise the barriers to entry so that no-one else could copy their ideas. Instead, they should have been worrying about how easily they could be asset-stripped when they went bust. These factors led the team to reject geographical incremental delivery, launching country by country, and instead to launch in many at once. This decision led to enormously increased complexity in the back-end of the business in order to sort out taxes, delivery and pricing.

Their conviction that the online magazine, fancy 3D graphics and virtual sales assistant Miss Boo, were the reason customers would shop at boo.com made it hard for them to split their product offering. When the first round of cuts axed these elements, the co-founder Kajsa Leander threatened to resign, convinced the company was cutting priorities, not 'extras'. With hindsight it may be easy to scoff at her conviction, but the fact remains that not all people see priorities (and thus what enters the first increment) the same way.

Boo.com envisaged a near-instant global dominance that would discourage competition, create a ready-made image of cool and could be extended swiftly to other market sectors. In giving way to this 'big win' temptation and rejecting incremental delivery, they lost everything. It's a reminder that business as a whole, as well as IT projects or software products, need to stay focused on delivering small, valuable increments early.

1.2. Benefits of delivering early and often

We've mentioned many of the benefits above, but here they are, presented in a nice tidy table:

Financial	Earlier profits Earlier cashflow Less investment
Marketing	Shorter planning horizon Earlier feedback from customers More reliable feedback Flexibility Image of consistent product improvement Customer lock-in
Engineering	Lower technical complexity Early field experience with technology Spreads technological commitments
Other	More motivating to teams Accelerated learning (Feedback)

Financial

This is the biggie. The most obvious benefit, and most intuitive, is that the sooner you start using the software (either intended to generate revenue directly or indirectly or make savings), then the sooner you will see a return on your investment. Quarter by quarter, this extra time in market can radically improve a business case, bringing in cashflow sooner (at a time when the cash is worth slightly more) and significantly reducing investment risk. Your revenue will be higher even over the same product lifecycle because it starts earlier. Your break-even point will come sooner and thus your ROI and other measures will look more attractive. Delivering any increment will help, but by delivering your most valuable increments first, you can really transform the overall value to the organisation (see the Prioritisation session for further discussion on this point).

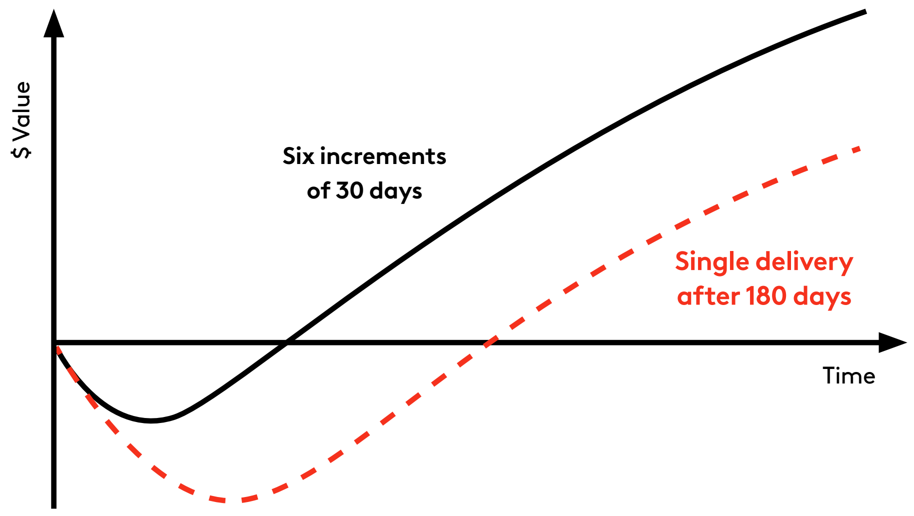


Figure 4. Earlier delivery of value through smaller increments

Perhaps you want to encourage your customers to receive their statements online, rather than printing and posting them out. You know that this will eventually save you several hundred thousand pounds per year. Migrating your total, highly complicated customer base online is going to be difficult. So perhaps you start with your most internet-engaged customers – just the 400 who signed up online to a direct debit payment scheme. This will still save you a bit of money.

You implement the increment and 100 customers ring up to complain that they hate the new system. You have learned something that has saved you a very costly mistake – if one in 4 of your total customers had got annoyed with you, the effects on your business might have been catastrophic. Now you need to rethink your strategy and customer motivation. How much should you invest in persuading your customers to change – perhaps you could offer them a 5% decrease in their bills. Does the business case still work in your favour? Once you fine tune your offer and get a success with your best customers, the business can observe the savings and decide whether it's worth rolling the project out to the next group. This is known as incremental funding – more money is allocated to a project as its success is proved.

The opposite would be to deliver all the value at the end of the project. Set up a completely online system and insist that customers either migrate to it or lose their account. This would be such a risky idea that few companies would do it. Instead, banks and utility companies try to entice customers to move to paper-free accounts with prize-draws, discounted fees or reminding customers of the environmental benefits – not by trumpeting how much it will save the company. Yet in software, we often see precisely this 'big-bang' implementation approach, with all the associated risk, potential cost and reduced value flow. Managers may be so focused on the economy of production cost that they cannot see beyond it, or they may be convinced that the dependencies within the project make it too difficult to break it down into increments - more on this later!

Increments also mean you know when to stop – the point at which you have gained all the value there is and everything on top just weighs the project down. As we will say many times in this course, many software features are unused. Sometimes known as 'gold-plating', sometimes referred to as YAGNI (you ain't gonna need it), these excess features represent waste. Put simply, they cost you money and time to add to a project and maintain. In return they don't offer sufficient value to justify their inclusion. But how do you know in advance what will turn out to be a killer feature and what will be a frill? Increments give you feedback early so you can kill the loser and invest in the winner.

Risk is an essential characteristic of any innovation, but it does not boil down to a project's ultimate success or failure. Other indirect financial benefits of incremental delivery include lower costs and lower risk associated with shorter lead times and lower inventory (such costs are lower in spite of a higher transaction cost). This is sometimes hard for people to accept because it seems so counter-intuitive. Consider the number of companies that centralise purchasing, for example, in order to drive down supplier costs and decide to order everything in bulk at the beginning of a project. This can end up causing higher costs later, when the 50 licenses for development software purchased at the start turn out to be a waste of money, because the developers now realise they need a different piece of software.

Marketing

The examples above remind you how financial benefits are tied to others. Early sales tend to equate to market share, which has a series of further benefits: establishing you as market leader; forcing competitors to play catch-up, and through frequent releases, giving a perception of your product as being continuously improved. By allowing you to fine-tune your offering to provide customers with the best value early, early delivery opens the possibility of locking customers into a relationship or charging more for your products.

Customer satisfaction may not show in immediate financial benefit to the company, but it is essential to future success. In the example of moving to paperless statements, incremental delivery permitted us to adapt to customer feedback – this controls the risk of annoying customers and allows you to please them sooner.

Engineering

Incremental delivery often enforces a degree of technical and architectural simplicity because it demands breaking systems into component parts. Even the most complex of databases must be built one section at a time and systems need to be decoupled to enable this. This can sometimes feel difficult to those dealing with legacy systems, but monolithic technical systems have their own problems – problems often ameliorated by a simpler 'lego-block' style of build. However daunting the task of decoupling the system or splitting it along the seams may feel, the benefits are likely to far outweigh the time required for doing it.

Incremental delivery allows teams to try out new technologies in a risk-limited manner. That might mean switching to a new programming language on a specific component or a new version of a database server on just one particular aspect of the system. While an additional piece of work might be required to link the new system to the old, by incremental delivery you can test the proposed functionality in a low-risk and low-cost manner while maintaining service on the old system.

Other

Knowing that you are adding real and measurable value is one of the most motivating experiences for a team. Seeing a customer using the feature, watching sales figures rise or complaints fall – such measures are a tangible expression of success – and success motivates more than a stirring speech from the boss or even a pay-rise. Knowing that the feature you are working on will go live next month (as opposed to in three years time when you may have left the company), is not only more exciting, but it is likely to force you to pay close attention to the quality of what you are writing. Together, these forces can significantly boost team performance.

Incremental delivery is driven through a feedback loop, normally because we use iterations to get us there. Feedback means learning, and since each piece of learning can be applied to the next iteration, the smaller the feedback loop the faster the learning process and the greater the chance of the painfully acquired knowledge being of benefit to the project and the wider organisation.

2 DELIVERING EARLY AND OFTEN IN ACTION

In 1969 the greatest decade in the history of mankind may have been coming to an end, but Buzz Aldrin and Neil Armstrong set foot on the moon, Concorde had its maiden flight and Boeing unveiled the Boeing 747, more affectionately known as the jumbo jet.

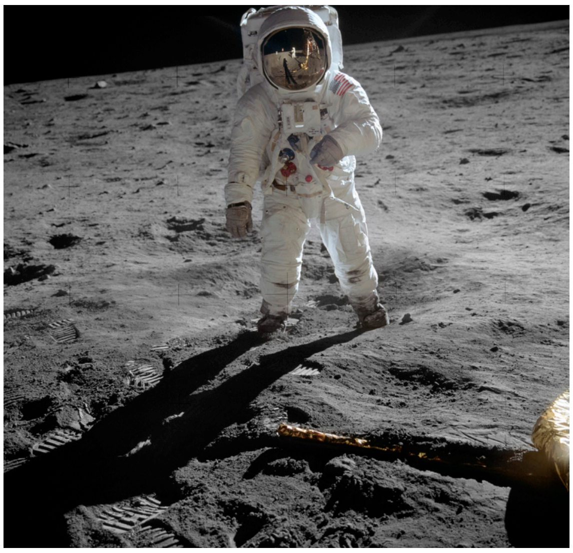


Figure 5. The first steps on the moon in 1969 – a feat not repeated after 1972

The last manned mission to the moon occurred in 1972 and Concorde was retired in 2003. Only Boeing 747s continue to go strong, with a total of 1,427 aeroplanes built by the end of 2011 and variants still flying today.

The aeroplane you climb onto today looks, from the outside, remarkably like the one you would have climbed onto in 1970. In more than 40 years, however, technology, regulation, passenger numbers, expectations – almost every external and environmental factor – has changed. The inside of the aeroplane and every component within it has changed too. Almost all of these changes happened incrementally – small functional elements, upgrades and changes made to the original design to adapt to new needs or expectations.

The original 747, for example, had an upper deck as a lounge area (with a rather attractive spiral staircase to reach it). Later this was reconfigured to become a first-class seating area, and several older aeroplanes were retro-fitted with the new layout. Range was extended on some models to improve the aeroplane's capacity to fly longer international route without greatly impacting on the number of passengers that could be carried, while other models received wingtip extensions to improve fuel efficiency. By the time the 747-400 series was launched, the number of dials gauges and knobs in the cockpit had reduced from 971 to 365 because of the use of electronics.

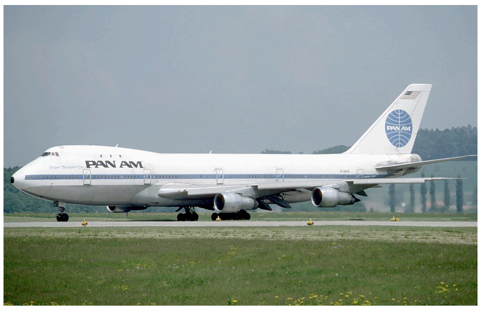


Figure 6. The original Boeing 747

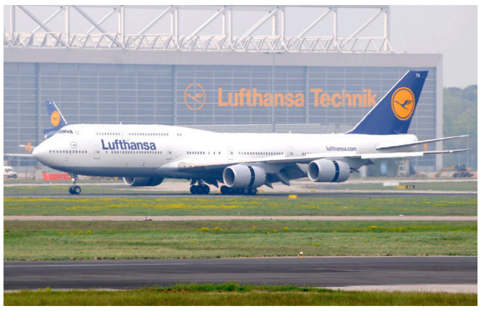


Figure 7. The latest Boeing 747

From design details of seating and fuel-tanks to production improvements including fuselage tooling or even expansion at the Everett plant where the aeroplanes are made, the Boeing 747 is an example of how incremental delivery can work. Each change provides value, but builds on the value of what went before.

CASE STUDY: Netflix

The digital distribution media provider, Netflix is reliant on software that works incrementally. It began with a breakthrough innovation, of course – renting DVDs via the internet and delivering them by post – but it was not alone in the market. In the UK, LoveFilm offered the same service, while Moviefilm and Quickflix also competed in the area, even the old rental-chain Blockbuster made a desperate last-ditch attempt to shift to an online model.

Netflix spent the first few years after launch building its customer base, experimenting incrementally with different subscription levels and changes to the website. The next increment was helping customers choose what to watch by providing film information and rankings. Now, the company has begun to move content to instant streaming to your TV, laptop or phone as an option likely to appeal to those with high connectivity. Netflix spends significant time on the design, guiding users through downloading in this manner, and providing linked suggestions ("if you liked this, you might like…").

If this is how the company has evolved at a macro level, at a micro level the increments are even more important – they test every new feature, updating and releasing the increment every two weeks and then discard anything that hasn't worked on the following cycle, two weeks later. Because they're simply testing one or two variables at a time, they can tell what's worked and what hasn't. It keeps them innovative because there's very little risk attached to failing, and it keeps them invested in the website as it is at the moment – a very motivating way of working for those who embrace the culture of change that comes along with it.

3 HOW DO I START DELIVERING EARLY AND OFTEN?

Higher value, lower risk, simpler engineering, happier customers and more motivated teams – so you're sold.

Now how do you go about splitting something into increments?

There are two elements to creating increments. The first is splitting a large and complex idea into smaller parts. The second is ensuring that the smaller parts have some kind of value in and of themselves.

Steve McConnell in his book Rapid Development suggests identifying 'a minimal subset of functionality that might be of use to the end-user'. In Agile Product Management With Scrum, Pichler recommends 'keep the visioning work to a minimum and quickly release a first product increment, or demo it to customers and users. Listen to the responses to see if you are shooting for the right goal. Then adapt.'

Neither of these authors tell you how you might identify the subset or work out what the first increment might be. Occasionally the subset is relatively easy to find. Suppose our customer wants to set up a review function for all the products on their website. How about we start with just his 10 most popular products and do the rest next week, or just permit a star rating now and allow comments the week after?

The example shows that there are often several ways to break down an idea (even if at first it seems difficult). You need to think about ways to break down the idea and then consider which of the ways is likely to be the most valuable to the customer. You might ask yourself what is the single most important benefit to the customer. The 'if-you-could-only-have-one-thing' feature, what would it be?

This is not always an easy question to answer. We frequently find developers, project managers, product owners and senior IT managers who announce: 'but my customer needs it all!' Many dependencies within a project are important – so important that it may genuinely be impossible to break all of them. But it is almost certainly possible to break a few of them.

3.1. Breaking down the big idea

We mentioned in the introduction that individuals and teams often sign up to the idea of incremental delivery only to fall at the first hurdle. Especially when in the middle of planning, it can feel very difficult to separate a product concept into independently valuable chunks. The benefits from doing so, however, are compelling.

Take the boo.com case study discussed previously. When the company was collapsing, boo.com realised that its distribution and shipping system could be a valuable property and they tried to sell it or license it to Deutsche Post and Kingfisher. Sadly for them, their recognition of a truly valuable increment came too late.

We're not pretending that decomposing a project into incremental steps is simple. It's not. It requires creativity, intelligent analysis and an understanding of what is of real value to customers and other stakeholders. What we are insisting is that, as Tom Gilb stresses in his article Decomposition Of Projects, 'this is a cultural problem, not a technological problem'.

3.2. Ways of slicing the cake

If you grew up in a large family, you know that there are many different ways to slice a cake – and for every way, you probably got expert at deciding how to end up with the largest slice…

Figure 8. One way of slicing a cake

There are many different ways of slicing up a big idea. Choosing between the different methods means that you must always keep in mind the overall objective: what is valuable to your customer. Only with this in mind it is possible to know which method of division is likely to be the most helpful.

Here are a few of the methods you might want to consider when considering how to slice the idea:

Value:

- What is the most financially valuable element of your idea? This might be directly revenue-producing, it might be something that saves money… Can this be delivered first?
- Don't forget that value is not only a financial measure – customer satisfaction is just as important, as may be the opinions of other stakeholders.
- Look for where the greatest value leads you. In Business-2-Business for example, electronic data interchange (EDI) might prove a far more valuable channel than creating a web-based application. Yet because the user interface of a web-based application is so much clearer it is often prioritised, despite being of lower value.

Risk:

- The riskiest elements of an idea are sometimes the ones to test first. Testing appetite for your product or the chance of success for a technology might help gain more funding to develop the idea further or to scale it.
- Eric Ries in his book The Lean Start-Up calls this the 'Build-Measure-Learn' loop, but he stresses that each increment should test a fundamental business hypothesis, not simply answer a technical question – e.g. would the customer pay more for a sharper picture on the TV? Not, does our HD technology provide 33 million pixels per image?

Stakeholder:

- Is there a specific stakeholder to whom you should deliver first, or whose views on what is most valuable should take priority? That might mean the piece of functionality your CEO wants to see, or it might mean launching to your top customer first. If you wish to create a product that functions for two departments, you might consider launching for just one first and adapting it later.
- Is there someone internal to your own business whose function mirrors that of an external customer? They might prove helpful in choosing the most important first increment.
- Is there a political element that can be balanced – i.e. if I do this for you first, then you'll do that for me? Be careful though, don't become internally focused because it's the customer's view that matters.
- Finally take a tip from Paul Graham of Viaweb who used to ask 'what would our competitor hate us to do most?' A competitor is a kind of negative stakeholder!

Urgency:

- Time is often a more important constraint than money. Can you divide by deadline? What is the most time-critical element of your idea?
- Don't forget to check that these are external deadlines. Internal ones tend to create false pressure.

Geography:

- Can you launch your idea to a more limited market? Facebook launched college by college through the USA and Canada before targeting high schools. Not until two years after it was incorporated did Facebook launch globally.
- This can work at several levels – launch to your home town before the country, launch to your top customer before anyone else…
- Which channel you launch through can help reduce the increment – selling online before creating a mobile-optimised site; creating an e-book before printing a physical version, or using a third-party supplier before developing your own capability.

Necessity:

- What can you continue using? In other words – what could be done last? If parts of the old system (however creaking and painful) can continue to work, then leave these elements to the end.
- What is the bare minimum that you can deliver; the element of which you can say 'without this, we're not in business at all'? This is sometimes known as the 'epicentre' and is often extremely simple. If you are selling a product, for example, your website might consist of simply a picture and an email to allow manual ordering.
- Is there a regulatory element to your project that must be delivered? This might have low economic value but be very important to your organisation's reputation.
- Many Service Oriented Architecture implementations end up as very slow non-incremental projects. Don't fall into this trap – salvage what you can of old systems or employ workarounds, while simultaneously working on an improved architecture.

Activity 2: Composing an increment using one consideration

This activity should take 30 minutes. You will use the results in Activity 3. You can complete the activity alone or as a team.

The Masterfoods board has issued a directive to the IT department to update all of the company's software systems (off the shelf) within a year. The budget is $2.5 million.

Masterfoods is the world's largest manufacturer and distributor of ice-cream dispensing units used in ice-cream vans, corner shops and cafes. The company has 45,000 employees split between the UK, the USA and India.

- UK: 20,000 employees (15,000 factories, 1,000 head office, 4,000 mobile sales)
- USA: 20,000 employees (18,000 factories, 500 satellite office, 1,500 mobile sales)
- India: 5,000 employees (4,500 factory, 500 satellite office)

IT did a similar project 5 years ago. It was a nightmare and cost nearly $8 million. Everyone is very anxious about a repetition of this disaster.

The company is growing and several IT managers have suggested that the systems should be packaged in such a way that deployment can be automated. Others have pointed out that one of the major stumbling blocks last time was that the hardware was so diverse, that many machines had to be upgraded before the new software could be deployed, while others had certain versions or drivers installed that made updates a lengthy process.

Licenses on legacy logistics software in the US factory are due to expire in one month. Support costs will increase by $50,000 per month.

A new European data privacy regulation requires sales systems to be compliant within 3 months. Fines of $20,000 per month will be imposed after that point and consumer groups are watching closely for any infractions. The Sales department have been reviewing software from a new vendor (which complies with the regulations), but they are a good 6 months away from signing a contract that covers all 5,500 reps. They are keen not to get tied in to contracts with the current vendor when they are not happy with the product.

The ERP system for the factories has been highly customised over the years. The system in India works in Tamil. The IT office in London has little or no experience of dealing with this, as it has all been done locally.

The board is promoting an initiative to cut staff at the US satellite office, by transferring the accounting and billing functions directly into the software so that invoices are automated and can be run from the head office in London. Projected savings are to cut 200 staff at an average salary of $30,000 each ($6 million a year).

A new initiative in India is to move from pure manufacturing supply of the US and UK to selling within India. A pilot team of mobile sales reps has shown promising results. The board wish to increase this to 500, which will mean providing them with the hardware and software the mobile sales teams use in UK and US (Electronic Proof of Delivery units with inbuilt ordering capacity). The projected cost is $300,000. Return in the first year is $200,000, expected to grow to $3 million within 3 years.

Consider ways to design increments based on the following:

- Value
- Risk
- Stakeholder
- Urgency
- Geography
- Necessity

This is NOT an exercise in prioritisation. Although it's hard, at the moment just concentrate on all the ways that you could split the project. You'll go on to evaluate these in Activity 4.

3.3. Valuable eating

IT people like to use food metaphors – their choice of culinary object might tell us something about developers and healthy eating patterns. Jeff Patton calls it a 'layer cake' (hence 'slicing the cake'). The software consultant Gojko Adzic prefers the idea of a burger to help teams break down ideas into increments.

As well as deciding which requirements we will place in the initial increment, often there are several ways to fulfil the selected requirements. Sometimes we can make do with a workaround, for example. We might eventually want to update contacts every few seconds, or send out automatic responses, but in the short-term we might be prepared to make do with a manual response, batched up nightly.

We will explore the technique in Activity 3.

Note that each 'bite' through the burger includes more or less functionality, enhancing the previous version. By making workarounds explicit, this method of planning can reassure the wider team that you haven't forgotten their requirements. Incremental delivery is intended to speed the feedback loop so that you learn and improve. In all but the most enormous projects, spending weeks deciding what should go into an increment defeats its own purpose. A visual plan can keep the process simple and still capture alternative options.

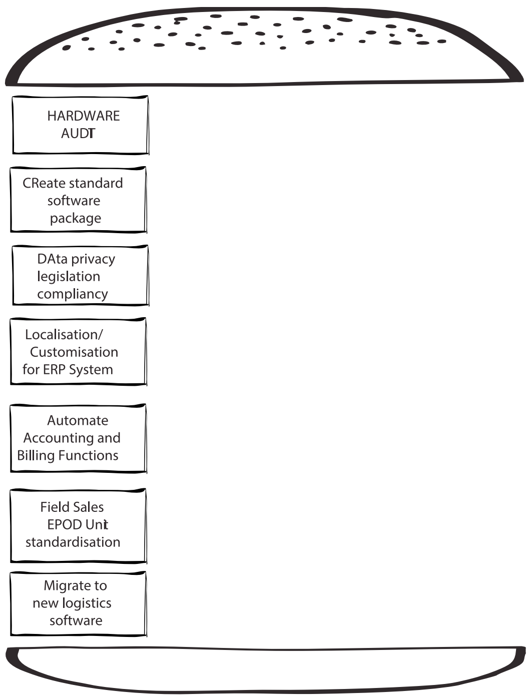
Activity 3: Designing an increment: composite composition

This activity should take 20 minutes. For best results we suggest working as a team.

Take your list of potential increment divisions from Activity 2. We suggest using the hamburger method to display these on a flipchart.

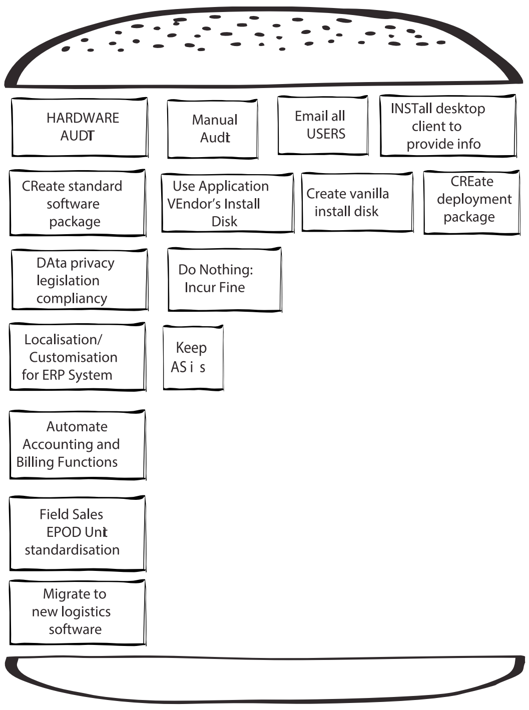
Task 1:

List all the different main sections of the project down the left hand side. Yours do not have to be the same as those that we have suggested as an example.


HARDWARE AUDT
CReate standard software package
DAta privacy legislation compliancy
Localisation/ Customisation for ERP System
Automate Accounting and Billing Functions
Field Sales EPOD Unt standardisation
Migrate to new logistics software

Task 2:

Now begin to write out the differing ways this could be achieved along the horizontal layer of the burger. We have filled in two example layers. Write your own as you discuss the problems.


HARDWARE AUDT
Manual Audt
Email all USERS
INSTall desktop client to provide info
CReate standard software package
Use Application VEndor's Install Disk
Create vanilla install disk
CREate deployment package
DAta privacy legislation compliancy
Do Nothing: Incur Fine
Localisation/ Customisation for ERP System
Keep AS i s
Automate Accounting and Billing Functions
Field Sales EPOD Unit standardisation
Migrate to new logistics software

Task 3:

Consider what you will take as your first bite. Use a colour highlighter to circle the tasks / quality level for your initial bite.

Try to ensure your increment is about 3 months long.

There is no right answer, but to see an example view of an appropriately balanced incremental plan, go to the Appendix.

3.4. When do you stop slicing?

Alastair Cockburn refers to a technique he uses as 'elephant carpaccio' – taking a big problem and slicing it into super-fine increments. He works with developers who produce increments that take only 15-30 minutes each. He disarmingly adds, 'That's not because we're so fast we can develop 100 times as fast as other people, but rather, we have trained ourselves to ask for end-user-visible functionality 100 times smaller than most other people.'

Such 'nano-increments' are easier to create on some projects than others. But you have to remember something very important – an increment requires a piece of functionality that is 'of value to the customer'. Sometimes people who are new to the techniques get so carried away that they slice too finely and in doing so lose the value.

Slicing into increments is not the same as dividing into tasks. If we want to warm the room then we might decide to build a fire. To do this we need to fetch kindling, logs and sweep out the fireplace. We need to build up the fire and light it with a match. Then we need to fan the flame and add logs until we have a nice roaring fire which heats up the room. Going to collect the logs is not an increment because it does not provide value towards 'making the room warm'. It is simply a task – albeit an important one. Going to fetch a jumper to keep us warm, while we build the fire, might be an initial increment.

The danger signals to look out for:

- Making the increment smaller feels pointless and frustrating
- The planning to slice the increment takes longer than doing the task
- The increment is small enough to be done by one person and requires no collaboration

Incremental delivery does not demand that you chop a system up impossibly – there are times when dependencies cannot be split. Just because a task can be made smaller it doesn't mean it is miraculously independent. Sometimes a large step is necessary at the beginning (the first iPhone was a major project). The point is not to add in more than is necessary – always strip back and then release new functionality in increments.

Activity 4: Creating carpaccio

In Activity 3, we designed an increment that we felt was around 3 months of work. Now let's look at how we could turn this chunk into carpaccio.

We wish to go as small as we can – start by asking yourself:

If this is what we could deliver in 3 months, what can we deliver in half the time?

Now halve the time again – what can you deliver?

Keep going until you reach the point past which you can no longer deliver a unit of value (what Paul Graham calls the quantum of utility). Don't give up too easily.

Plus remember, although there are unquestionably dependencies in this problem, upgrading even a single computer is valuable.

CASE STUDY: Y Combinator Funding

Paul Graham made his money, and a name for himself, when he sold his e-commerce software Viaweb to Yahoo! for $50 million. Since then he has become almost equally well know for his entertaining essays on everything from why nerds have a hard time at school to the superiority of particular programming languages. He has also begun a venture capital investment business, looking for new or current graduates with business ideas.

Twice a year he runs a start-up 'bootcamp'. This attracts numerous applicants for the 40 places. The bootcamp runs for three months in Silicon Valley and provides seed funding and advisors to develop an idea during an intensive period. By the end of this period, each founder is expected to have a working product and a solid pitch to make to investors on 'Demo Day'.

The point of the seed funding and three month development period is not to polish up a business plan or make useful contacts or plan the branding (or any of the kind of things boo.com might have spent it on), but to deliver working software. Of course, each 'larval' piece of software also has a larval business model and an idea of how it should be presented to users and investors. It is a complete first increment – neither just the software that might drive a business, nor the business concept that requires a technology platform to be built, but a complete, working combination of both.

Thus on Demo Day, when 400 investors pack in to watch a day of pitches, the start-ups are looking to generate serious leads which they will close in the next few weeks to provide the funding they need to move their business to the next stage. Some start-ups will launch before the Demo Day, at the initial prototype meeting (after 2 weeks) or at one of the weekly mini demos. As Paul Graham writes, 'Usually we advise startups to launch when they've built something with a quantum of utility—when they've built something sufficiently better than existing options that at least some users would say "I'm glad this appeared, because now I can finally do x."'

In fact, that's a pretty good definition of the first increment for both start-up and established business – 'quantum of utility'.

3.5. Customer journeys: deciding on what goes into an increment

Occasionally a project comes ready divided into distinct features. Our example about Masterfoods had some clear requirements. Once you'd decided how these could be divided up, you needed to think about prioritisation as well. More often, however, a product exists as an amorphous mass of ideas – some vague, some definite. Your job might be to split these down into smaller increments to assist developers working on micro-functions, but equally it might be to bundle several items up into a releasable increment that will provide real end-user benefit to the customer.

To return to Alastair Cockburn, he readily acknowledges that micro-increments without an overview of the whole projects can be counter-productive.

The slices of carpaccio can be thought of as small user stories, but the idea of the animal needs a larger vision, one that will ensure the thin slices fit together to form the right shaped animal. When dealing with software we often refer to this larger idea as a customer journey or story map – a strand of functionality made of several different features that will provide real use and value.

For example, your customer may only ever use the online banking system to check her balance, but there still needs to be an element of design for the home page, security needs to be assured, the link to retrieve the data must work… Understanding these links in the customer journey is vital. You can't simply ask your customer to pick the most valuable function and then hand it to a developer.

We should also probably stress that this process can look rather different depending upon whether you are in an established business where you are looking at what to put into the next feature release, versus a start-up new concept where you want to create a core set of functionality that will allow you to test the heart of the idea.

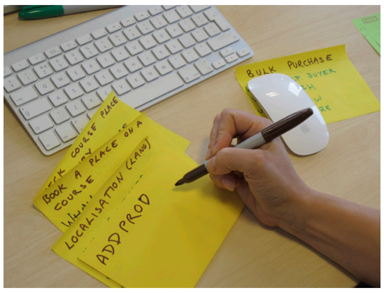
In either case, we recommend story mapping to ensure you are filling your increments with the right features.

Activity 5: Story mapping

This extended activity uses a worked example to aid comprehension. We took our actual project 'Build the VFQ website', but we suggest that you take a project you are working on (or considering working on) and use that. Do this activity with your team. Set aside 2 hours for Part 1, and then return to it later when you have walked other stakeholders or users through the story map to gather their comments. In total, we expect the activity to take a minimum of 3 hours. It may take longer if you wish to calculate development time estimates.

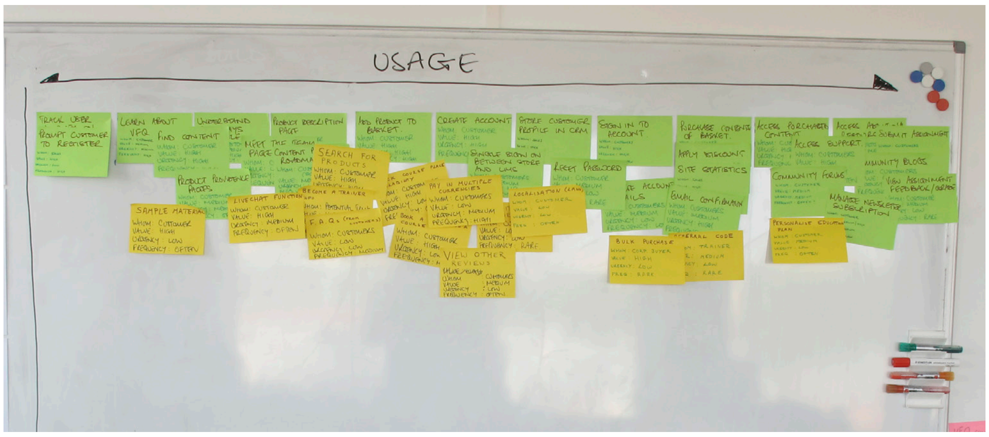
Part 1

Task 1: Begin by collecting as many features as possible: things that your software idea could do which you think people will find useful (some of these may not be about the customer, they may include back-end operations as well). Write each feature on an index card or sticky note.

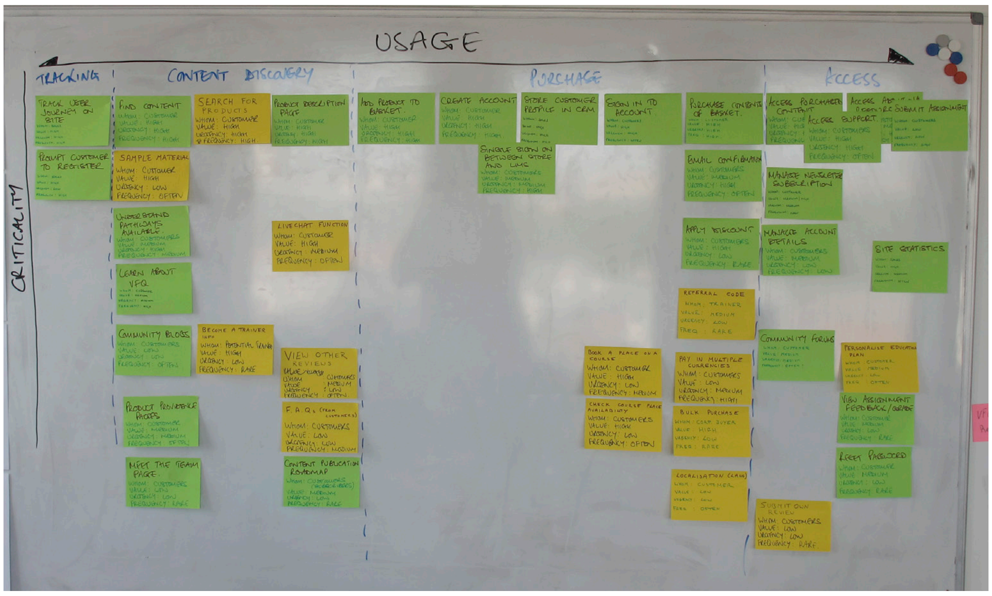


Task 2: Below the brief description, add the person or role to whom the activity is valuable. On separate lines, write the likely value and the urgency. Although in reality you might have an idea of a numerical value, for now simply use a label low, medium or high.

Task 3: Draw a set of axes on a large sheet of paper or a whiteboard. The x-axis signifies usage order and the y-axis suggests how critical each activity is from low to high. Take your cards and begin to order them horizontally in their usage sequence, overlapping where necessary.



Task 4: Now separate the cards vertically, according to how critical they are, but keeping them in their sequence. Draw dividing lines that relate to business processes, logical breaks in the workflow. Jeff Patton points out in his article on the process of story mapping that where your features have a new user (person or role to whom the feature is valuable) this is often a natural break in the process.



Part 2

Task 5: Share your initial model with a wider group – this may include business analysts, team leads from related projects, project managers, a senior executive, as many users as possible.

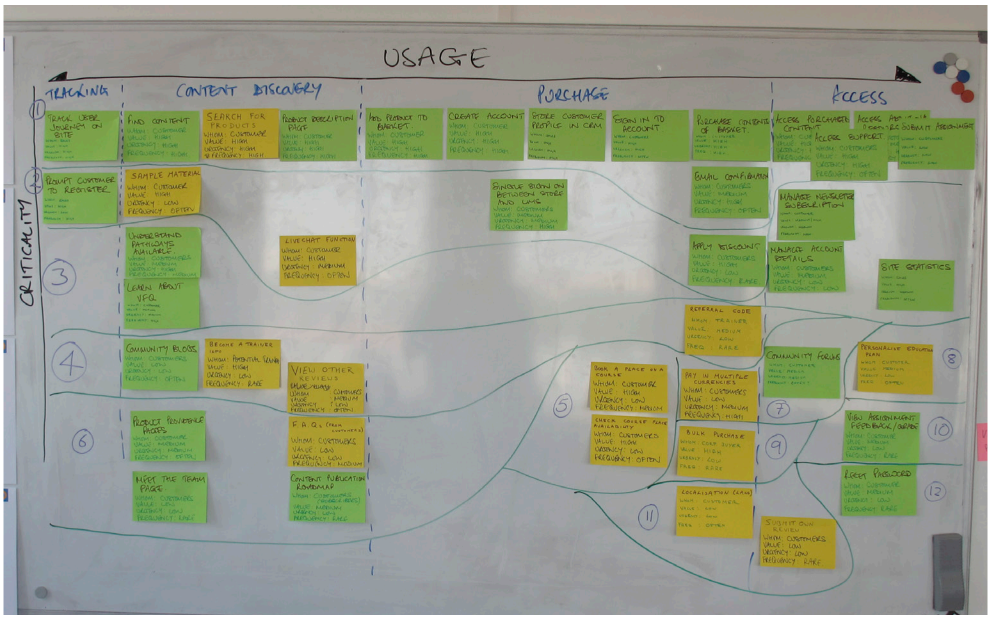
What business processes are missing? What roles might be missing and what might those people's needs be? Talking to these people is likely to throw up a large number of extra features. Write them up and slot them in.

Task 6: Divide the map of features using horizontal lines to define your increments. Jeff Patton calls this step a 'system span', a horizontal slice that cuts through the business processes from start to finish. Since this will form our first increment, we are looking for the smallest set of features which will be useful in a business context: 'the bare bones minimum anyone could legitimately do and still use the system'.

To improve your plan, at this point you can ask the team how long it would take to deliver this first section. Try and provide some actual estimates for each feature.

Task 7: Now you have your first increment, consider what is missed out. Are there any dependencies missing? Who will not be able to do what they want and what workaround (paper or software) might exist that they could use in the short term?

For example, in our plan, although Applying a Discount was an important feature, we left it out of the first increment, since we were able to apply it manually to orders when necessary.



Now consider an estimated release date. That is, enough features that you should be able to produce something in 14 or 30 days, or whatever time-box you select.

Commentary:

Jeff Patton summarises the benefits of the story mapping exercise as: ‘Because you’ve arranged features in sequential order, you now understand what features depend on one another. Because you’ve arranged them by criticality, the important features are now emphasised at the top of the plan. Because you’ve divided the features into business processes, you have a better idea of the functionality that supports each major business process in your software. You have determined the minimal feature span that lets you get your system up and running, end to end, as soon as possible.’ All this information is provided in one convenient picture. By employing a little common sense, we should be able to carve off the smallest possible releases that will still be useful to the people who ultimately receive those releases.

3.6. Enabling delivering early and often

Choosing the iteration length

The iteration length needs to be long enough to permit a piece of end-to-end functionality to be built. If the team don’t have enough time to create a subset of the product then they will fail to iterate - they will simply be randomly dividing their time up. On the other hand if they are unable to create a sub-set in 4 weeks, then they are probably not working hard enough to split ideas down into small increments. In general, the more risky or unknowns surround the product, environment or technology then the more frequently the team should seek feedback. In general, most teams work to iterations of between 1-4 weeks.

The walking skeleton

In software, a temporary solution is often known as the 'walking skeleton': just enough end-to-end architecture to permit a small element of functionality. For example, in an online retail site, the walking skeleton would permit a single user to enter the shop, place an item in a basket and pay. This connects all the major elements together, long before there is any list of items to buy, user interface or even data capture. Further function and architecture can then be fleshed out. Alastair Cockburn defines it as, 'a tiny implementation of the system that performs a small end-to-end function… The architecture and the functionality can then evolve in parallel.' In other words the focus must be on something working, even if it works inelegantly.

When German troops cut off the entire British Expeditionary Force at Dunkirk, the government calculated that it had two days in which it hoped to rescue 30,000 British troops. Luckily for them, the rescue operations continued for 7 days, eventually rescuing over 338,000 troops. The main reason for this unexpected success was the famous 'flotilla of little ships'. Boats which would have been quite incapable normally of crossing the channel were guided across by the Navy, and ferried men from the shallow waters of the beaches and harbour at low tide to the waiting destroyers. This emergency and 'inadequate' solution, ended up rescuing ten times more men than would have been possible using 'correct' transport.

Continuous integration

Huge strides can be made in incremental delivery without changing anything about the underlying architecture or way you work, but one thing that helps is if your team is in the habit of integrating frequently. This enables frequent deployment – because you know that any software which has been written, is literally ready to go as a working version.

As David Farley and Jez Humble write in their book Continuous Delivery: 'When this practice is followed, then the software is always in a working state. If your tests are sufficiently comprehensive and you are running tests on a sufficiently production-like environment, then the software is in fact always in a releasable state.'

Of course it is tempting to ignore this advice, because it feels faster to 'branch the source code' and make changes to the branch. The risk is that the challenge of integrating this back in at the end proves far more disruptive and difficult than any time saving made by not having to keep the application working throughout the changes. Farley and Humble stress, 'If other teams are working in the meantime, the merge at the end can be hard – and the bigger the change, the harder it will be. The bigger the apparent reason to branch, the more you shouldn't branch.'

Moving to continuous delivery

It is not an enormous step from continuous integration with incremental delivery to making several very small releases per day – a practice now known as continuous delivery.

Adrian Cockcroft describes how the process works at Netflix, where there is no need for developers building new code to request a change to a production system, or extra capacity in advance. Instead, they use a web-based portal to deploy any change, running the new code alongside the old. They put a single 'canary' into traffic and check its performance. If all goes well, the same developer moves all the traffic to the new code and the old version is deleted some time later. If there were problems with the canary, or if problems develop with the volume of traffic, all the traffic is instantly routed back to the previous version. Control is devolved to the teams themselves, who decide their own dependencies and manage security, as well as having a series of alerts and tools to help them spot if anything goes wrong and then to fix it themselves.

Another version of this is sometimes known as blue/green releasing, intended to helpfully distinguish it from red/green releasing, because its primary characteristic is that there is no stop/start. Two identical production environments run side by side. While the blue environment is live, new code can be uploaded onto the green environment checking that everything is working. Then, at a point the team choose, traffic can be switched over. There is no downtime and if anything does go wrong, then traffic can just be switched back, making it easier to rollback errors.

CASE STUDY: Facebook's canaries

Every day, Facebook engineers push millions of lines of code for fixes, features and improvements onto their site. This is part of what users come to Facebook for – they can do more stuff on it now than was possible last year. It's part of why advertisers and businesses come to Facebook – this is a company that works out new ways to help them get to customers. In fact, Facebook relies on constant change.

The problem is that change is one of the best ways to get disruption, outages or problems. Naturally with 600+ million users relying on Facebook for their social lives and, increasingly, business advertising – it's very important that the changes don't do that. Facebook outage! Internet implodes! It's happened, and the newspaper headlines were appalling.

Not unnaturally, that means that Facebook pays quite a lot of attention to how it manages its releases. Chuck Rossi, a release engineer for Facebook, describes the steps the company takes. He stresses that the first and most important is not a tool or a process, but a cultural difference. Developers who write code are responsible for the performance of that code from start to finish. They're not allowed to rely on a tester to find the bugs in it, or a release engineer to make sure it doesn't break the build. A developer is responsible up to and including the point where a user starts playing with the changes on the site.

Given how large the company is (30 or 40 development teams), the infrastructure is heavily automated and uses IRC Bots. The system even demands the presence of the specific developer before his or her revision will be merged to go out in the push. This backs up the culture of responsibility – if you're not watching your revision go through the tests and go out then it won't go. Even worse, if your revision has messed up the deployment then there's a 'down thumb icon' that marks you as a slightly riskier developer. It's information that will be shared in your performance review and while one or two are fine – they happen to everyone – a trend is a bad indication.

They have a release schedule that works around a pattern connected to actual use (only very small changes go out on a Friday, for example, to make sure that when huge traffic arrives on the weekend the change doesn't cause problems). The main push occurs on a Tuesday. This means that all developers commit their code to the main trunk throughout the previous week. On a Sunday this collection of changes is cut as a branch, tested on Monday and sent out on Tuesday. Smaller daily pushes happen throughout the rest of the time, from fixes to slightly riskier ideas on a Wednesday.

The push itself uses a type of canary testing as a three-part process:

- First the build goes to A1 – a small set of production boxes to which only employees are routed.
- If the A1 deployment looks good, the build goes to A2, a "couple of thousand" boxes to which only a small percentage of users are routed. This can be done on a sliding scale using a Gatekeeper to bump it up to 1% of users, take it back down to 0 and check all the results before gradually sliding up.
- A1 and A2 are like canaries in a coal mine – if a problem is discovered at these stages, the build goes no further. Only when no problems occur is the build promoted to A3 – full deployment.

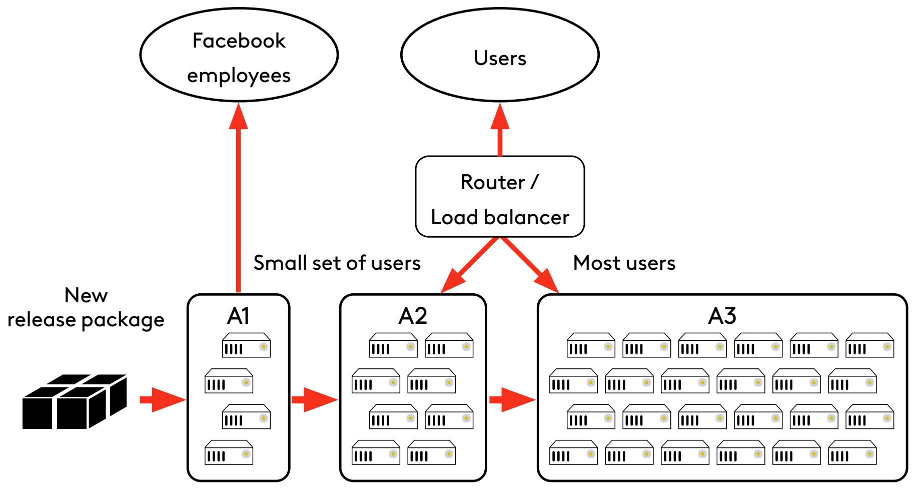


Figure 9. Facebook's canary testing as a three-part process

4 THE DISADVANTAGES OF DELIVERING EARLY AND OFTEN

Before anyone gets over-excited by the heading, let's state unambiguously – we think delivering early and often is a good idea and that its benefits outweigh any potential disadvantages and costs. However, you need to discover what level is right for your business.

Activity 6: Breaking down your increments

This activity will take 30 minutes. For the best results we suggest this as a group activity but it can be done alone.

During the first 15 minutes of the activity, review a project that is either currently underway or has recently completed. Identify the first increment and the features of which it was composed.

Could the increment have been smaller? Identify the elements that were not strictly necessary to a minimal end-to-end piece of functionality.

What were the obstacles or organisational impediments that led to the increment being larger than necessary?

During the second half of the activity, discuss the impediments. The following questions and ideas might help:

- Does the architecture of your software allow small components to be released, or is this a constraint? Could you refactor the code? Would you need to persuade others of the value of this? Could you build a financial case for automation or reviewing the architecture?
- Are there assumptions about what the customer will accept? Can these be tested? Can you find a smaller element which could be demonstrated to the customer?
- Is there a back office requirement? Is it financially not viable to provide launch or marketing support for a new product until it reaches critical mass? Again, can an element be tested or released to a limited customer base? Don't forget that Facebook test new ideas internally and to top users.
- Is there a cultural bias in favour of large projects? Who might be able to break this? What arguments are most likely to persuade management to trial a small project?

What steps could you take to make a change that would enable future projects to design smaller increments?

Do you need to involve others to make these changes? If so, you may want to take them through the results of our next activity (Activity 7). After that suggest the changes and try to enlist their support.

4.1. Transaction cost

Incremental delivery can have a higher transaction cost than a big bang implementation. Let's imagine that you are having the windows in your house changed to double-glazing. The salesman tells you that if you order all the windows at once, there will be a significant discount on the cost per window. Also, you will only have to pay a single fitting charge. If you order the windows one at a time, not only will your materials cost rise, but so will the labour charge, because you must cover the window-fitter driving to and from your property several times.

It makes perfect sense to have all your windows changed at once. It makes so much sense that almost no-one considers doing it differently.

But just for a moment imagine that instead of owning a house, you owned an entire block of flats. The contract is even larger, the discount the company is prepared to offer you for doing the whole lot even more tempting. But the risk is also much greater. There are plenty of horror stories of cowboy builders who fit the windows badly, or use substandard materials, and by the time the owners complain, the fitters have vanished into insolvency.

In this case, it is almost certainly worth swallowing the higher transaction cost and having one flat's windows changed first. You can then gather feedback – how much disruption did the builders cause, how well did they fit the windows, how good is the heat retention? Having considered this feedback, you can go on to award the contract for the whole building. If you decide not to proceed you have learned something important relatively cheaply.

Software development is even more uncertain than double-glazing. The benefits of risk management, feedback and early value to the customer more than outweigh any associated costs. There are also other savings – like those mentioned in the Netflix example. When fully embraced, incremental delivery means less time lost in meetings to plan deployment, less time fixing bugs and lower overheads as teams lower transaction costs through intelligent use of tools and automation.

Nonetheless, these benefits don't all magically appear immediately. Managers are often more focused on transaction costs, which they feel immediately, rather than future efficiencies about which they aren't confident. Having incredibly short iterations might mean using up too much of your time in planning, demonstration and review meetings - a high time transaction cost. The level that works for you will depend on how innovative or risky your project is and thus how often you need to seek feedback. The trick is to begin with a small increment and prove the benefits – an increment of incremental delivery you might say…

4.2. Marketing overload

If your whole organisation is geared towards a particular release cadence, it can be very hard to change this. Let's imagine that you release a new model of food processor every year at the annual kitchen appliances show. Your retailers are used to this – the show is two months before their annual re-merchandising. Your marketing team is used to this – they produce a beautiful looking brochure, they contact the retail buyers, they have the stand at the show booked. The factory is used to this – they invest in new tooling once a year. The customer service desk is used to this – they know when old models will no longer be supported and how to train staff accordingly. When you suggest flowing a new change to the existing model every 3 months, you create uproar.

Software is a lot more malleable than hardware, but departments can still be very wedded to a release schedule to which they are accustomed. There are two answers to this problem – the first is that a delivery schedule does not have to be the same as a release schedule. You can arrange releases to suit your own marketing purposes. Microsoft launches a major new upgrade of a programme like Excel according to its own backroom convenience, but it combines this with upgrades delivered dynamically. The second is to embrace the benefits of frequent delivery. Service companies tend to thrive best on a continuous relationship with their customers, constantly improving the service and providing instant reactions or fixes to customer feedback. But to take advantage of these benefits, companies often need to transform the way they test, receive feedback and manage customer relationships.

4.3. Technical breakthrough

Some breakthroughs require big changes powered by large batches. You may need to keep a new technology under wraps in order to stop your competitors copying it until you can launch a game-changing product – many technology companies are famous for the paranoid secrecy that surrounds their latest project. Small steps may keep things working, but evolution can feel boring compared to revolution.

Once again, the real brilliance comes in knowing what is the minimum you can launch with – even if that minimum is pretty big. The original Kindle didn't have a search function, the iPhone didn't do MMS texting and Google Docs launched using only its own form of word processing and spreadsheets.

CASE STUDY: Rapid release proved disappointing for Firefox users

Mozilla's internet browser Firefox which proudly describes itself as 'different by design' and insists 'doing good is part of our code', made a change to its delivery system, updating with new features, fixing bugs and making other changes every 6 weeks. They expected users to be delighted, they also decided that this would stop users changing over to the newly launched Google Chrome, which delivered speedily. Now as soon as Chrome launched a feature, the Firefox team could match it and push it out to its users.

But there was a problem. Lots of users hated it. The main problem was that the update process was intrusive. A series of dialog boxes asked customers to restart the browser. This was irritating and time-consuming. Moreover, the changes often stopped supporting add-ons or broke compatibilities with other software. Even 'improvements' to the user interface called forth irritation. As the blogger Jono points out, 'there's no UI better than the one you know'. Even if the product was becoming theoretically better, to existing users it was annoying. In fact, Firefox users began defecting to Google Chrome in worrying numbers – the very outcome which regular release had been designed to avoid.

Google Chrome had launched with what blogger and Mozilla employee, Jono, describes as 'an empty box of a browser ... wrapped around an excellent updating system'. That meant that Google Chrome users were hardly aware of updates. They just appeared and the Google team could then tweet about them to make interested users aware. Those who were passionate about the product enjoyed this, those who just wanted it to do the same basic job they'd been using it for previously could ignore new features.

The moral might seem to be that it's important to plan how you do updates; that they should be unobtrusive and low impact on the user. That's certainly important, but there is a second significant point – not all 'improvements' work for all users. Changes to the user interface may be great for new users but require tiresome re-learning for existing users; cutting a feature used by only 1% of your users may make economic sense to you, but if those 1% are vocal or high-spenders or early-adopters, then there may be other downsides. No hard and fast rule like 'release early and often' can take the place of real knowledge of your customers, intelligent analysis and a short feedback loop. In short, it's yet another example of why software cannot be separated out from business disciplines of customer understanding and response. Mozilla should have worked out that the update process was annoying customers and taken steps to deal with this much earlier than they did.

5 CONCLUSION

Just how valuable can incremental delivery be for your organisation?

It is possible to predict the benefits of an incremental delivery approach using a formula, somewhat unfortunately known as the BADD formula.

The formula derives from work done by Emergn with the Danish shipping company Maersk Line. It complements work in the Prioritisation session on how incremental delivery can improve return on investment. It forms a compelling argument to present to the most transaction-cost-averse, agile-suspicious, bottom-line-focused CFO imaginable.

This formula tells you absolutely nothing about the tricky question of how to split your big idea into increments, it simply demonstrates why you might want to.

> What extra benefit would we expect if we divided our original increment size (Io) into a number of smaller increments of equal length (In), each of which is released as soon as it is delivered?

$$\% \text{ Benefit Increase} = 1/2\,(1 - In/Io)$$

In = New Increment Size (usually measured as a unit of time)

Io = Original Increment Size

The current increment size is 180 days (that is, software is released after 180 days) and the new projected increment size is 90 days (that is, we increase our release frequency as we reduce our increment size). Applying the formula, we get the following results:

$In/Io = 90/180 = 0.5$

then $\% \text{ Benefit Increase} = (1/2\,(1 - 0.5)) = 0.25$

This means there is a 25 % Increase in Benefits.

Assumptions:

- Value is equally distributed.
- Each smaller increment is released when ready.

The formula holds true no matter how many increments you add, up to the theoretical limit of zero increment size.

Activity 7: Applying the BADD formula

This activity is designed to help you provide some real figures to show how valuable incremental delivery might be to you. This can provide an argument as to why it might be worth expending effort, time and even money to achieve it.

The activity should take 45 minutes.

For a project that has either recently started, or is about to start, we can model the results from differently sized increments.

Before applying the formula, you should identify an average increment size. We suggest measuring the size in days or weeks as appropriate. Select at least 3 past projects that have delivered within the last 12 months and identify the increments they delivered. Calculate the time for each of those increments and then determine the average. It will help if the projects you select are easily comparable to the current one on which you are applying the formula.

Look at the plan for your current project. When do you plan to release the first increment? Compare this to the average release for your past projects – you should consider what will happen if issues cause your first release date to slip, as well as if you manage to decrease it.

Using the BADD formula, calculate:

1. What will happen if the first increment increases (perhaps to be equal to your past projects).
2. Benefits if you halve the time until the first increment is delivered.
3. Benefits if you drop cycle time to a quarter.

With these numbers to hand, discuss the benefits with those people involved in the project. Is there anything that they could do to reduce the size of their first increment? How might you be able to help them?

Naturally, any formula contains assumptions. In this case, it assumes that we can bring value forward and that value is released as a linear proportion of total value. In other words, we are not assuming that we do the highest value first, although we are making assumptions about both costs and value.

As the creators of the formula comment, 'It almost becomes impossible when applying this information not to consider breaking down work.'

In other words, no matter how painful or difficult the process may seem, it is worth doing. When you add to this the benefits of prioritising your increments to deliver the most valuable first, you can see why success in this area can transform your project and potentially your entire organisation. Financial value is not the only measure of success, and we have discussed other potential benefits and disadvantages in this session, but it would be a rare organisation that felt financial value was unimportant.

Learning outcomes

Now you have completed this session you will understand:

How increments and iterations are defined

- One or a number of iterations combine to produce an increment
- An increment is a usable, valuable subset of functionality
- Iteration involves seeking feedback and thus emphasises validating design, rather than creating something perfect and complete

Appreciate the benefits of delivering early and often

- Financial – delivering earlier means more time in market, which should translate to more sales (or more savings). This can transform the business case, with a quicker break-even point and improved cashflow. Non-performing projects can be killed faster, lowering risk
- Marketing – more time in market should also mean a greater awareness and higher market share. Early results and feedback allow the team to fine tune the product ensuring subsequent releases please the customer more
- Engineering – incremental delivery enforces greater simplicity of architecture, breaking down system dependency and complexity. This in turn permits the test of new technologies in a lower risk manner
- Other – a sense of urgency and immediacy of results motivates the team and encourages a focus on improvement

Identify examples of where incremental delivery is used as a successful business model

- Incremental delivery does not only work with software – it can be used in businesses from traditional engineering to retail
- Software is uniquely well-adapted to it, being malleable and where parts can have value independently of the whole

Begin to break projects into increments through:

- Developing a mindset to split dependencies and ideas
    - However hard it may feel to split a project into increments, it is almost always possible
- Using different prisms to split ideas
    - Value, risk, stakeholder, urgency, geography, necessity

Know how small to go with your increment

- What is the minimum functionality, the 'quantum of usability', that allows you to test an idea or which is of value to your customer

Be able to map a customer journey or story strand to create slices or bites of functionality

- A practical activity designed to lead you through the process of identifying elements of functionality by urgency and value

Put in place practices that enable delivering early and often

- Setting the length of the iteration between 1-4 weeks
- The Walking Skeleton - the minimal architecture that connects the minimal element of end-to-end functionality
- Continuous integration – an increment needs to be 'done' and ready to release – that means fully integrated with both internal and external systems
- Continuous deployment – consider blue-green releasing as a way to reduce risk and deploy faster

Balance potential drawbacks or limits to delivering early and often including transaction cost and technological breakthrough

- Incremental delivery does cost more, but overall transaction costs can lower as the practices that support it (including automation and shorter planning horizons) free up more resource
- Separating back-room delivery from marketable release can be done to suit the organisation's marketing requirements – but this will undercut many of the financial benefits of incremental delivery
- Technical breakthrough – in a few cases the rarity of a new property may favour the big bang rather than incremental delivery – the trick is to still strip out excess functionality in order to launch fast

Demonstrate the value of incremental delivery

- Use the BADD formula to convince waverers of the potential economic benefit

BIBLIOGRAPHY

Adzic, G., 2012. Splitting User Stories: The Hamburger Method. Gojko Adzic: Building Software That Matters, [blog] 23 January. Available at: <http://gojko.net/2012/01/23/splitting-user-stories-the-hamburger-method/>. [Accessed 24 April 2012].

Agile Alliance. Guide to Agile Practices: Incremental Development. [online] Available at: <http://guide.agilealliance.org/guide/incremental.html>. [Accessed 13 April 2012].

Airliners.net. The Boeing 747-100 & 200. [online] Available at: <>. [Accessed 23 April 2012].

Anderson, D., 2003. Agile Management For Software Engineering: Applying The Theory Of Constraints For Business Results. Prentice Hall.

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

Boeing. 747 Program Milestones. [online] Available at: <http://www.boeing.com/commercial/747family/pf/pf_milestones.html>. [Accessed 23 April 2012].

Boeing. History: 747 Commercial Transport. [online] Available at: <http://www.boeing.com/history/boeing/747.html>. [Accessed 19 April 2012].

Boeing. The Boeing 747 Classics. [online] Available at: <http://www.boeing.com/commercial/747family/pf/pf_classic_back.html>. [Accessed 23 April 2012].

Brown, T., 2009. Change by Design: How Design Thinking Transforms Organizations And Inspires Innovation. Harper Collins.

Cockburn, A., 2008. Elephant Carpaccio. [online] Available at: < http://alistair.cockburn.us/Elephant+carpaccio>. [Accessed 23 April 2012].

Cockburn, A., 2007. Incremental versus Iterative Development. [online] Available at: <http://alistair.cockburn.us/Incremental+versus+iterative+development>. [Accessed 13 April 2012].

Cockburn, A., 1996. Walking Skeleton. [online] Available at: <http://alistair.cockburn.us/Walking+skeleton>. [Accessed 24 April 2012].

Cockcroft, A., 2012. Ops, DevOps and PaaS (NoOps) at Netflix. Adrian Cockcroft's Blog, [blog] 19 March. Available at: <http://perfcap.blogspot.co.uk/2012/03/ops-devops-and-noops-at-netflix.html>. [Accessed 20 April 2012].

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

Dolman-Darrall, P., 2012. The BADD Formula: A Business Case For Agile. Value, Flow, Quality, [blog] 6 March, Available at: <http://www.valueflowquality.com/the-badd-law/>. [Accessed 17 April 2012].

Gilb, T., 2008. Decomposition Of Projects: How To Design Small Incremental Steps. [online] Available at: <www.gilb.com/tiki-download_file.php?fileId=41>. [Accessed 23 April 2012].

Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases Through Build, Test, And Deployment Automation. Addison Wesley.

Jono, 2012. Follow-up To The Firefox Updates Post. Evil Brain Jono's Natural Log, [blog 11 July]. Available at: <http://evilbrainjono.net/blog?permalink=1097>. [Accessed 27 September 2012].

Krebs, J., 2008. Agile Portfolio Management. Microsoft Press.

Larman, C., 2003. Agile & Iterative Development: A Manager's Guide. Addison Wesley.

Louis, T., 2000. Boo.com Goes Bust. <tnl.net>, [blog] 19 May. Available at: <http://www.tnl.net/blog/2000/05/19/boocom-goes-bust/>. [Accessed 27 September 2012].

Manifesto for Agile Software Development, 2001. Principles Behind The Agile Manifesto. [online] Available at <http://agilemanifesto.org/principles.html>. [Accessed 18 April 2012].

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

Patton, J., 2008. Don't Know What I Want, But I Know How To Get It. Agile Product Design, [blog] 21 January. Available at: <http://agileproductdesign.com/blog/dont_know_what_i_want.html>. [Accessed 20 April 2012].

Patton, J., 2005. It's All In How You Slice. [online] Available at: <http://www.agileproductdesign.com/writing/how_you_slice_it.pdf>. [Accessed 23 April 2012].

Pichler, R., 2010. Agile Product Management With Scrum: Creating Products That Customers Love. Addison Wesley.

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

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

Schwaber, K., Sutherland, J., 2012. Software in 30 Days. John Wiley & Sons.

Scott, A., 2012. Continuous Delivery And Ruining The User Experience. WatirMelon, A 93% Watir Based Blog, [blog 22 July]. Available at: <http://watirmelon.com/category/continuous-delivery/>. [Accessed 27 September].

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

Siegler, M., 2011. Snoozing And Losing: A Blockbuster Failure. TechCrunch, [blog] 6 April. Available at: <http://techcrunch.com/2011/04/06/make-it-a-blockbuster-night/>. [Accessed 20 April 2012].

Siegler, M., 2011. The Next 6 Months Worth Of Features Are In Facebook's Code Right Now (But We Can't See). TechCrunch, [blog] 30 May. Available at: <http://techcrunch.com/2011/05/30/facebook-source-code/>. [Accessed 27 September 2012].

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

User Interface Engineering, 2006. The Freedom Of Fast Iterations: How Netflix Designs A Winning Web Site. [online] Available at: <http://www.uie.com/articles/fast_iterations/>. [Accessed 19 April 2012].

Wikipedia. Boeing 747. [online] Available at: <http://en.wikipedia.org/wiki/Boeing_747>. [Accessed 20 April 2012].

Wikipedia. History Of Aviation. [online] Available at: <http://en.wikipedia.org/wiki/Aviation_history>. [Accessed 20 April 2012].

Wikipedia. Iteration. [online] Available at: <http://en.wikipedia.org/wiki/Iteration>. [Accessed 19 April 2012].

Wikipedia. Iterative and Incremental Development. [online] Available at: <http://en.wikipedia.org/wiki/Iterative_and_incremental_development>. [Accessed 13 April 2012].

APPENDIX

Activities 2 and 4 were an exercise regarding incremental delivery, not prioritisation, but we have written a suggested plan. It's not the only solution, just one of many. Our purpose is to illustrate the kind of thinking and balancing that is required when examining what should go into each increment.

Value:

There are two direct costs to avoid: increased support costs in the US and regulatory fines in the UK.

There is a potential saving in the US from an ERP system upgrade and there is potential revenue in India from new hardware and associated software.

A major potential cost is over-run on the IT budget. Past experience suggests this could be serious.

Risk:

Bad PR if we ignore regulation and pay the fines.

Risk that deployment could be derailed by awkward customisations and localisation (Tamil in India), or by old hardware. This could threaten both schedule and budget.

New mobile hardware may not be suitable for our reps.

Stakeholders:

Board – reduce overhead and increase revenue.

Sales – already invested in the new vendor.

Urgency:

External dates (fines and increased costs) give these a measure of urgency, while the company's cashflow and board pressure demands IT shows an ROI fairly quickly.

Geography:

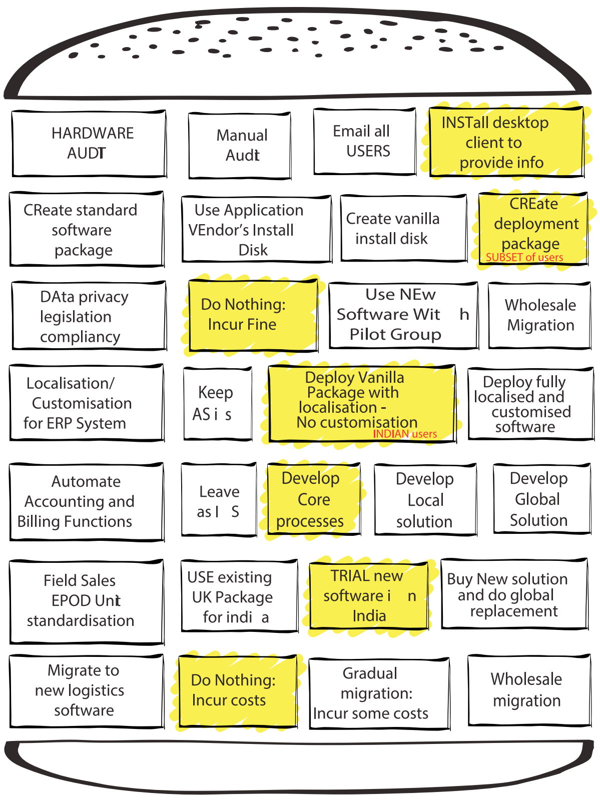
Could split along country lines since there is a degree of localised IT support, English is our largest market.

Necessity:

Automating the deployment is important in order to keep costs under control.

Rationale:

We want our increment to be small so that we can gain valuable feedback to learn about areas we are worried about and lower our risk of the project going overtime and overbudget. We also want to deliver some value early. To this end, we designed our first increment as follows.


HARDWARE AUDT
Manual Audt
Email all USERS
INSTall desktop client to provide info
CReate standard software package
Use Application VEndor's Install Disk
Create vanilla install disk
CREate deployment package
SUBSET of users
DAta privacy legislation compliancy
Do Nothing: Incur Fine
Use NEw Software Wit h Pilot Group
Wholesale Migration
Localisation/ Customisation for ERP System
Keep AS i s
Deploy Vanilla Package with localisation - No customisation
INDIAN users
Deploy fully localised and customised software
Automate Accounting and Billing Functions
Leave as I S
Develop Core processes
Develop Local solution
Develop Global Solution
Field Sales EPOD Unt standardisation
USE existing UK Package for indi a
TRIAL new software i n India
Buy New solution and do global replacement
Migrate to new logistics software
Do Nothing: Incur costs
Gradual migration: Incur some costs
Wholesale migration

Discover how many people in the organisation have outdated hardware that must be replaced – these form a representative sample throughout all business areas and countries. We will learn valuable information about this to design the rest of our increments.

Using this group as our initial increment, we identify the requirements of a vanilla desktop operating system which will be packaged and deployed automatically. We will deal with the localisation issue in India as part of this, but not the customisation problems since we will be providing everyone with the same basic desktop package. In this increment we also wish to test a thin slice of the end-to-end accounting and billing automisation. Within our hardware group, we expect to find 2 or 3 users in the US ready to retire or leave naturally and not be replaced, as we test this.

We also plan to ask the new Sales vendor to supply hardware and software to test with the group of sales reps in India.

We will pay the increased costs of software support and fines in the short term, taking the risk of negative PR and trying to counteract it.

Assumptions and calculations:

Our first increment will take 3 months. Our second increment another 3.

800 people have old hardware and that these will be spread evenly throughout the company and countries.

The rate of savings and revenue has a constant distribution. That is – at the end of increment 1 we will begin to save on 3 people's salaries for 9 months ($67,500). We will also make revenue from the India team ($150,000).

We will incur fines for 3 months ($60,000).

We will incur increased support costs for 6 months ($300,000).

We will incur the full cost of hardware and software for the India team ($300,000).

That means our net loss will be $442,500. However, we are taking on this cost because we are convinced that the learning which will enable us to keep the system upgrade to the $2.5 million budget is more valuable. Consider the previous overspend of nearly $5.5 million. The loss is a mere 9% of this risk.

In increment 2 we plan to move to the new vendor, thus complying with regulation and take a further step towards completely automated deployment. While doing so we will complete roll-out to the Indian office and factory. We will investigate alternative solutions to upgrading the logistics software in the US, such as contract negotiations, but we do not expect to be able to be rid of this cost until increment 3.

We have not designed further than this because we plan to use feedback from the first two to decide on what should go into the next increment, although obviously we intend to avoid further costs on software support as soon as possible. Several decisions, such as whether to move to the new Sales vendor or how to proceed with automated deployment depend entirely upon results from increment 1.

VFQ Value Flow Quality

valueflowquality.com

emergn.com

by emergn