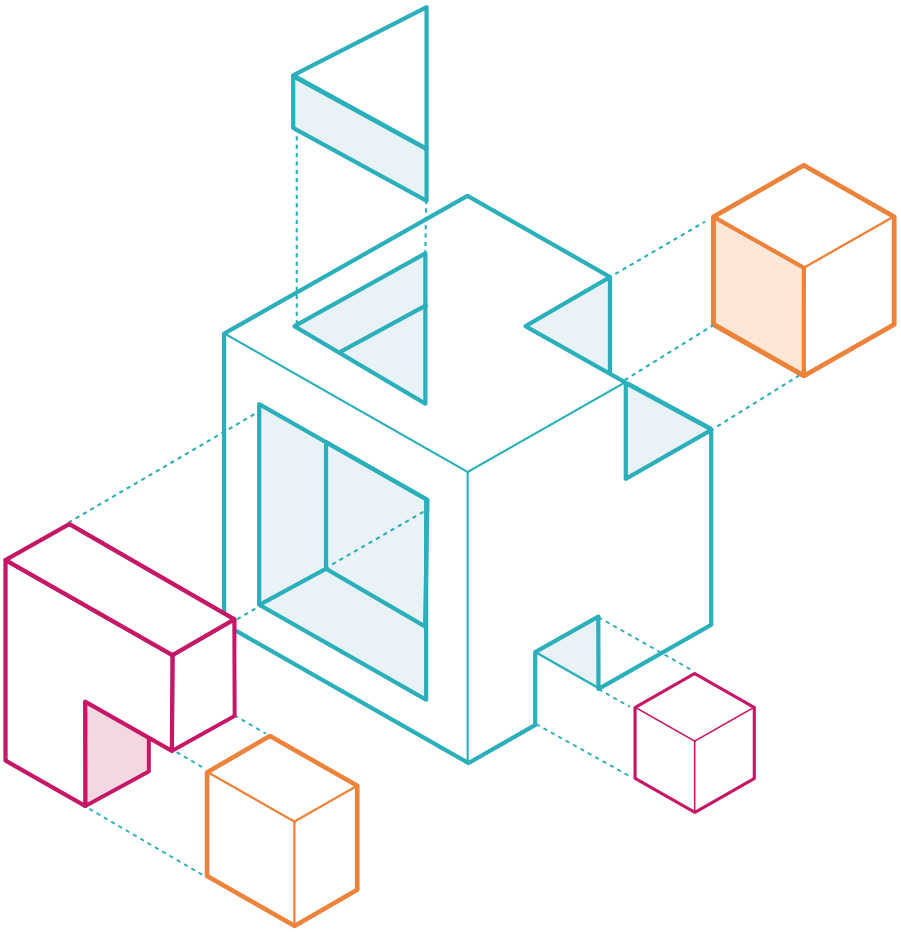
# Think small to deliver big

## Approaching design and delivery in small batches

Delivering software is hard, there's no doubt about it. And that fact is often, if not always, underestimated. Even setting aside internal or external dependencies, legacy or technology constraints, known unknowns – or the infamous unknown unknowns – it's often made exponentially harder by teams with good intentions trying to design the perfect solution long before a single line of code is written. In this paper, we look at how to address this paradox, exploring the benefits of starting small to ultimately build big; why this is difficult to do in practice, how to identify opportunities to break down bigger problems, and practical approaches to avoid failure.

**A white paper by Rob Smith and Simon Walder**



SCOTT LOGIC

ALTOGETHER SMARTER

# Contents

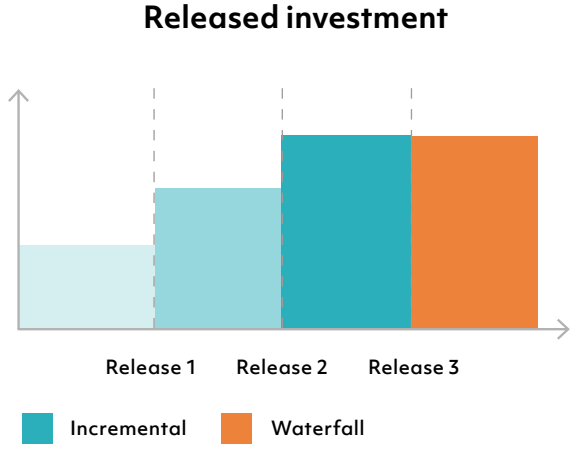
# Introduction

**IT projects, like engineering projects, were traditionally delivered in large batches following a Waterfall approach. Before the advent of DevOps automation, long, complex and manually intensive test and release cycles dictated that systems be changed as infrequently as possible. Therefore, to minimise and eliminate the cost of change it was important to "get it right first time". An admirable, albeit impossible, aspiration.**

In this context, Waterfall and other similar project management variants aimed to understand the whole problem before delivering the solution, ideally in a single large batch, removing all uncertainty from the process before embarking on the next sequential phase of the project, and incurring the release overhead only once.

However, large batches generate numerous issues, and in recent years a number of new approaches have risen to prominence based on small-batch delivery. These were explicitly designed to avoid the problems of their predecessors and provide the opportunity to experiment and learn, reduce lead times, and avoid scope bloat.

In theory, delivering in small batches and working in an Agile way provides a significantly improved return on investment (ROI) based on early and regular release of value, and enables teams to home in on the simplest possible solution to meet the user need. However, evidence suggests that although things have improved significantly over recent years, many projects are still drawn into traditional, destructive big-batch thinking.




There are many reasons why this may be the case, but anecdotally it appears that project teams struggle with the decomposition of problems, and – as a direct consequence – more specifically with uncertainty resolution and incremental delivery.

As a consultancy, we work with our clients to help them solve their problems and are invariably approached with a comprehensive solution already in mind. This is perfectly understandable for a variety of reasons: often it's a problem that is already present and has been discussed internally many, many times; nothing comes for free, and usually there's a requirement to specify the output in order to secure a budget; and ultimately, people just like to solve problems.

Breaking this habit and starting small simplifies the task at hand and allows for both direct and indirect value to be realised quickly. Direct value being, for example, working software that resolves the problem in incremental steps. Indirect value being the removal of uncertainty, gaining new insights, and potentially ripping something up and starting again. The key is to understand that you are on a journey and not to overly focus on the destination. You should focus instead on the steps on this journey, how you plan to realise value, and, importantly, what value looks like.

This paper looks at the challenge of delivering in small steps. It looks at resolution of uncertainty and at incremental delivery, and provides guidance on practical approaches to achieving the desired outcomes. It also looks at the specific challenges of the minimum viable product (which, in the wrong hands, simply becomes itself a large batch) and system replacement (intuitively always a large batch).

# Starting small... it's not as easy as it sounds

**IT projects continue to fail – regardless of improvements in technology, our understanding and appreciation of it, improvements in software development practices, tooling designed to make things simpler, and agile methodologies like Scrum.**

Google is awash with advice and articles espousing the virtue of Cloud, DevOps, Agile; on how such things can help organisations succeed in their digital transformations – and with good reason, as there's some solid reasoning and evidence behind them all. However, the application of these new services and ways of working does not automatically lead to success. The problem? It's in the mindset – the way you think about the problem an IT project is intended to solve.

Thinking small to deliver big is the mindset you need, and it's not as straightforward as it sounds.

As problems are discussed over time, usually with an ever-growing audience, they tend to expand rather than contract. As the problem grows, the perceived benefit of resolving everything up-front grows, aided and abetted by a belief that specifying the complete solution will help identify big-batch budget efficiencies. All discussion, whether directly relevant or not, promotes learning and exploration, so we are not dismissing it as wrong or a waste of time. However, the risk is that all of this up-front discussion of the problem can also generate and reinforce preconceived ideas that are hard to shake, resulting in a solution that is difficult and inefficient to deliver.

Often problems in one area of an organisation are reflected in another, prompting the inference that there is benefit in solving both together. However, the old adage 'A bird in the hand is worth two in the bush' could not be more relevant; while problems might appear to be the same, they are often subtly different, with each area having different processes, data, KPIs and drivers. With each area feeding in their subtly different requirements – that must all be met – the size, scope and complexity of the problem grows.

Invariably, this overcomplication leads to rounds of meetings and discussions. Other problems come to the fore, and often the same people are involved in resolving them. The discovery phase bloats, people become spread thin, and distractions mount; decisions take longer to make, which further compounds delays. This is a self-perpetuating vicious cycle.

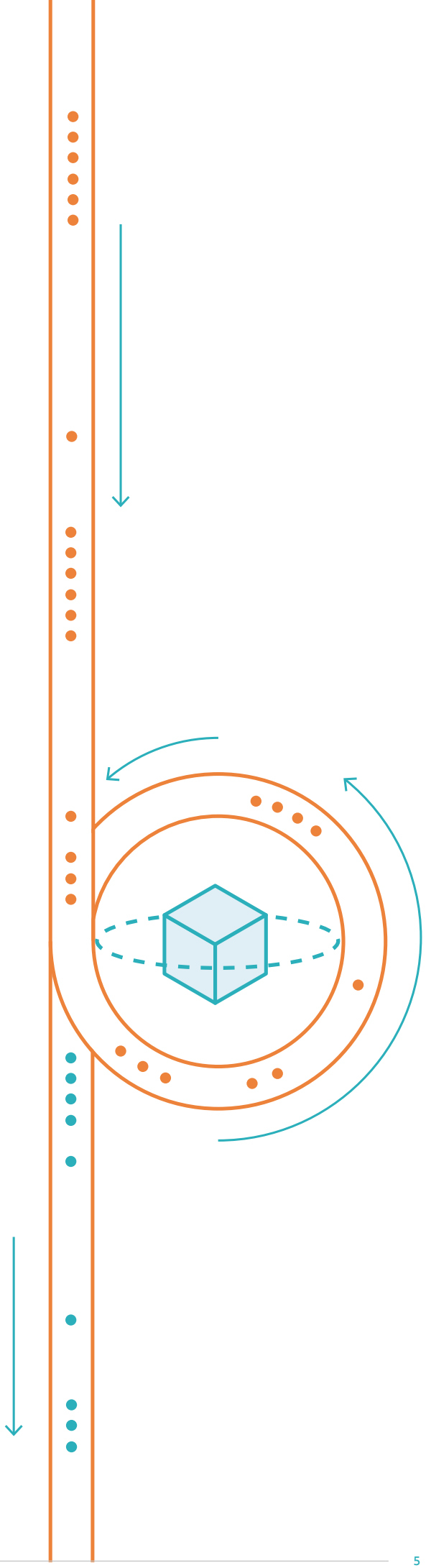
As time passes, the impact on the organisation intensifies. The need to resolve the remaining uncertainties mounts and delivery pressure is applied, with original deadlines being cited. The problem has been discussed at length, and so there is a widespread perception that it must be well understood; this perception increases the pressure to get delivery underway, leading to poor decision-making.

As a problem grows, so do the number of dependencies. These can come in many forms, from softer dependencies such as people's time, to harder ones such as the platform to be used. Each of these dependencies has its own needs that can cause drag, delays and sometimes halt things altogether.

Of course, one of the main reasons why overcomplication happens is simple human nature.

People like to be People Pleasers. This can be for a variety of reasons: being the good corporate citizen ("If we do this, then it will solve problems for all"); or the fear of saying no ("I don't want to be seen as obstructive or negative or not a team player"); or being seen as the person who can deliver ("I'll deliver this, and everyone will be happy (with me)").

People like to solve problems and design solutions. This is especially true of those who have a wealth of experience; it's very easy to draw parallels with previous problems and solutions and dive right in. Experience should absolutely be used to inform the design of a solution – but not by trying to find a shortcut to the destination. Instead, it's vital to follow the steps of the journey, starting small to deliver big.



# So, what can you do?

**Starting small does not mean that you ignore the entirety of the problem. Nor does it mean that you avoid speaking to end users or the wider business to really understand the problem, or that you steer clear of discussing possible solutions.**

What starting small means is that you focus on the journey over the solution itself: you break down the problem into smaller steps; you look at how those steps fit together and in what order; you quantify the value of each; and you understand the dependencies, how to remove them, and when they could be safely reintroduced.

Essential is an appreciation that there are no wrong answers, only new insights to gain. This sounds dangerous and expensive; but if your steps are small, then so is the risk and cost.

## Understanding the journey

It's easy in theory to talk about the journey to a solution, but the route seldom seems simple at the outset. Before you get started, it is essential to define what ultimate success looks like, with a clear understanding of value.

Once in the iterative cycle of implementation, it can be relatively straightforward to understand the value of a particular release. But early in the project, this can be less intuitive.

The problem during discovery is where to start and how much work is required before commencing implementation. Focusing on absolute value is difficult, as many of the questions that need to be answered don't appear to directly correlate to value. "How do we improve customer service" is a little abstract when the question at hand is focused on technology selection, architectural choices or other similar uncertainties.

Large-batch thinking provides clear guidance on when and how to resolve design uncertainty (i.e. at the beginning), but delivery in small batches allows many design activities to take place continually throughout the project. This is one of the benefits of Agile: that it provides additional options for uncertainty resolution beyond mere analysis, options that encourage learning and shorten lead times.

Nevertheless, some key uncertainties must still be addressed at the start, even when delivering incrementally, before they become major issues. While it may be easy to refactor a method, it is unlikely to be as easy to make changes during delivery to the architecture, technology platform or UX design.

## Addressing uncertainty

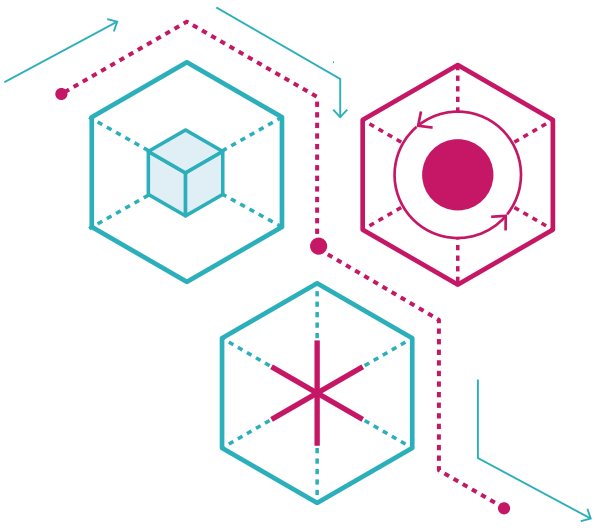
Value is accrued during Discovery through the resolution of sufficient uncertainty to enable the project to move ahead into implementation with the confidence that it won't be derailed at a later date by a showstopper; particularly one that should have been addressed earlier.

To provide focus to the team and optimise the journey, Discovery can be approached in short iterations. It is essential not to overanalyse, and using 'design sprints' should help. Each iteration should be focused on a specific outcome: 'deciding on technology', 'establishing a deployment pipeline' or 'completing user research of initial flow'. Where necessary, these can further be subdivided to give greater focus e.g. technology performance, scalability, operational overhead, training needs, and so on.

For example, you may need to: undertake user research or identify appropriate technology; build a web interface, an API or data store; source or build a platform for hosting. Throughout, consider the following:

Throughout, consider the following:

- What needs further discussion and how do we drive those discussions?
- What will be challenging?
- What will slow us down and stop us learning?
- What can be done right now and what can be deferred to later?
- Is the work necessary as an up-front activity, or can it be undertaken as part of the incremental journey?
- When will the work be completed?
- What does the logical order of these steps look like?



## Wider horizons through iteration

A government department client of ours was investigating how citizen information could be easily shared between new digital services and legacy systems via an Event Notification Service (ENS). The client had already identified user needs, explored constraints, measured legacy on-boarding lead times, and produced a conceptual architecture.

We were commissioned to deliver a proof of concept (PoC) in six weeks to demonstrate the viability of the department's conceptual architecture and intended technology choices – choices based on preconceived ideas of how the ENS would work. But it quickly became clear to us that a much more flexible, extensible and valuable solution was available to them.

In collaboration with the client, we went on an iterative design journey punctuated by regular demonstrations of working software – which prompted discussion around which areas to investigate next – which we again demoed, prompting further discussion and further investigation. So excited was the client by progress that they extended our engagement by three weeks to investigate even further.

From being a PoC around a specific technology choice, the iterative journey supplied the client with the design for a technology-agnostic, self-service ENS with the potential to provide far more value to the department than their preconceived solution had ever envisaged.

# Incremental delivery

**Once the key uncertainties are resolved and there is a clear understanding of the project foundations, it's time to think about the journey through to the solution.**

Incremental delivery refers to the approach by which a system is delivered in small batches ordered according to business value. Each batch should be complete and testable and deliver value (ideally) independent of other batches. In an ideal world, analysis and design are undertaken as part of the process and the architecture is allowed to emerge, but we don't live in an ideal world and this is seldom a sensible option.

The theory, and ideally the practice, of incremental delivery is to start with the simplest possible solution and to layer on complexity. The ruthless focus on value and simplicity should ensure that the optimum solution is developed for the investment, avoiding the nasty problem of scope bloat associated with Waterfall. However, perhaps more importantly, incremental delivery provides a mechanism with which to break down big complex problems into smaller batches.

## A focus on value

Devising an incremental strategy requires a continual focus on value. It is important to have an idea of the ultimate solution, but, as discussed earlier, the journey is more important than the destination; and of course, the destination may change along the way as ideas are implemented and tested, and learning is applied.

By focusing on value, it is usually possible to fragment a solution by audience (data) or function, or both in combination. Asking the question, "Where will the maximum value accrue?" will typically lead to a specific subset of the user base or a part of the business process, which can then be implemented as an incremental step.

Each increment should offer clear value in itself, but it may not be possible to accrue that value without combining it with others to create a releasable increment.

It may also be necessary to include functionality or "workarounds" that will not form part of the final solution. Examples of this include manual processes and integrations with old systems to facilitate parallel running.

For example, a previous client was launching a new vehicle insurance product, and this is a broad, multifaceted problem; the underwriting rules alone become exponentially complicated as demographic information and vehicle options are added. However, from an incremental perspective there are several options, the obvious one being to look at the largest demographic group (i.e. the one containing the most potential customers); but other, potentially better, options exist.

Launching a new product always represents a risk, and therefore a more appropriate strategy may be to look at the easiest, least costly increment to implement: the demographic/vehicle type combination with the least complex underwriting rules. In this way, it may be possible to launch the product in a very short time to test the market and justify further investment.

In the insurance example, the insurer launched a product for over 25s with no motoring convictions, who were driving low insurance group commercial vehicles – commercial vehicle insurance being the new specialist insurance product. When the product was launched, the website provided a ring-back facility to anyone who fell outside of the restricted criteria, providing all users with a "complete" customer journey and allowing the call centre to upsell other products.

Discussed in more detail later, the MVP in this instance was of course very small and was deployed quickly at a fraction of the expected investment. Once live, this provided valuable feedback on usage demographics and buyer behaviours, which were incorporated into the prioritisation of further increments (over 20s, multi-driver, heavier vehicles etc.).

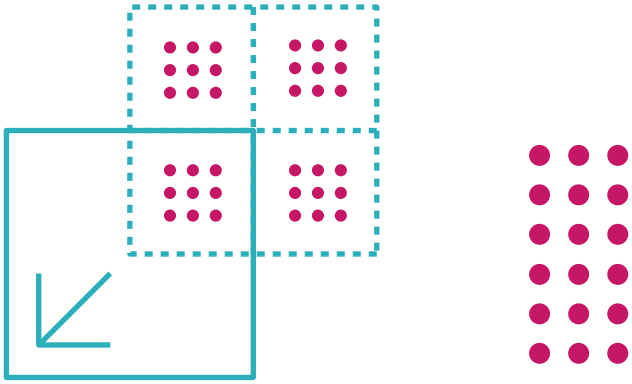
This also provides an example of leveraging existing processes to enable systems to be deployed before they are complete, providing an early return on investment and reducing overall risk. Here the balance is between the cost of the "workaround" (the call-back facility) and the value of the early launch.

## An example from retail

A number of years ago, one of this paper's authors was engaged by a retailer at very short notice. They were planning a new product launch a month hence, but had been beaten to market by a competitor with a very similar offering. Their challenge was how to launch as quickly as possible before the market share was significantly eroded.

When the project was reviewed, it became apparent that there were two major blockers: the functionality to prevent fraud, and the integration of availability checking in the retailer's point of sale system. Focusing on value, it became apparent that launching the product was an order of magnitude more valuable than the potential cost of fraud (working using standard fraud metrics). Moreover, it was possible to provide an in-store tool to check availability, which – although more time-consuming for sales staff – was not a sufficient reason to stop go-live.

The new product was launched within a week, with the additional functionality added as later increments. The market share was captured and the return on investment of the interim solutions was in the region of 20x.



# Fostering delivery success

The new mindset will greatly increase the potential for your project to be successful, but it is not a silver bullet. Here is further advice on ways you can foster delivery success.

## Demonstrating something drives conversation

A key benefit of the incremental approach is feedback and learning. A picture paints a thousand words and the sooner you can put something in front of users, problem owners or stakeholders, the better. Taking a single step or simple requirement and turning it into something that can be demonstrated drives learning and conversation.

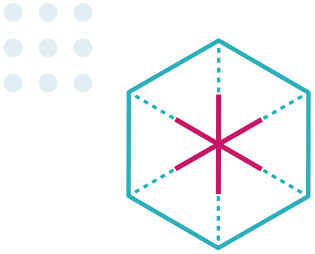
The team learns from doing it. During the build, they ask questions and seek clarifications, not just from external sources but from each other. This fosters a shared understanding and good working relationships.

The users, problem owners and stakeholders see how their requirements have been translated by the team. This promotes discussion on what has been built and demonstrated, leading to new insights and a better understanding which informs decision-making around future increments. Sometimes, it can lead to a tangential discussion, helping to identify an adjacent opportunity.

## Tips and tricks – avoiding the traps

### Keep things simple

As previously mentioned, focusing on the destination lends itself to a preconceived solution, overcomplication and delays. Keeping things simple promotes agility and the rapid realisation of value. Most importantly, it allows you to deliver working software, and drive conversation and learning.



### Do one thing at a time

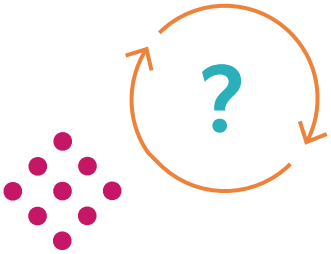
In the modern world and workplace, it's easy to get distracted and disappear down a rabbit hole, especially when the distraction has merit. It's important to capture these distractions, but agree to look at them later. Focus on the task at hand and prioritise the distractions as part of the next round of planning.

## Stop trying to do things perfectly

There is a time and place for perfect (or as reasonably close as is sensible to attain), especially as a solution begins to mature. Nobody wants to continually revisit work due to issues with quality or operability. At the beginning, things just need to be 'good enough' – the learning from delivering a thing and the discussion facilitated through demonstration holds greater value than it being production ready. (However, this is not an excuse for sloppy work or ignoring best practice!)

## Stop trying to please everyone

Successful delivery involves a certain amount of compromise and pragmatism, but sometimes an external agenda may be to the detriment of that success. Accept that you will need to make decisions that someone won't agree with. Don't be dismissive, and do capture. "We won't do this now, but let's discuss and maybe we can include it later."




## Challenge everything

Ask simplifying questions every day. It's easy to get caught up in the conversation, on the benefits reaped once the solution is delivered, and how that solution will look. Care must be taken to keep thinking about the steps on the journey. When things are proposed, decisions are being made or planning is in progress, ask yourself the following questions::

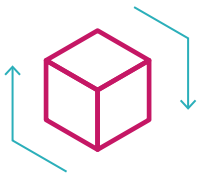
- **Is a solution being proposed without understanding the journey?**
- **Do we need this? And if so, do we need it right now?**
- **What is the short term and long term impact of postponing, and conversely if we do look at it now, what's the benefit?**

## Tactical vs strategic

Beware, here be dragons.

By starting small, you are making an active decision to demonstrate quickly and often, to promote learning and discussion, and to build iteratively towards incremental delivery. This can mean choosing one direction or technology over another because it's easy, simple and quick (Tactical). Nothing comes for free, and pros and cons need to be weighed up; "easy, simple and quick" may offer short-term benefits, but not be the right strategic decision for a business.

If you are lucky, the choice between tactical vs. strategic can be made quickly and by an individual, or small agreeable cohort. On average, this is not the case and can be a minefield to navigate. Part of any discussion here should be the benefit of maintaining momentum and how that outweighs waiting on a decision, or the yearning to discover a panacea. Once you have started, it's often more expensive to have a team spinning wheels rather than learning and resolving uncertainties to inform decisions. The question is "What can we do now to keep moving without making a future change prohibitively expensive?"



# Incremental thinking and the danger of the MVP

**The minimum viable product (MVP) is a great idea: avoid scope bloat and aim to launch the simplest solution possible that delivers value. However, the execution of the MVP often defeats the aim.**

In these cases, the release is viewed at the outset as a single batch, and therefore subject to many of the issues experienced in Waterfall. This is particularly true when implementing replacement systems, which is discussed below.

The problem in thinking about an MVP is very similar to that of Waterfall delivery in that the solution very quickly becomes a proxy for value, and the focus turns to the functionality the product "should" contain. Customers of the system see the MVP as a "single bite" and aim to ensure that the bite includes everything that they believe they have the appetite for.

Another common MVP-related issue is differing understanding of what the term means in a particular context. One person's anticipated MVP can differ quite considerably from another's, and if one is the customer and the other the supplier this can lead to significant frustration on both sides. This, of course, can easily be addressed by the development of a detailed specification, but that's precisely what we're trying to avoid.

The solution is similar to that described above. The incremental journey, which focuses on simplicity throughout, will naturally provide an optimal journey to the optimal MVP. Importantly, the journey towards the MVP is more important than the destination – if each incremental step is correct and in its simplest form, then the MVP will naturally be reached and recognised as part of the process. Also, it is much easier to answer the question, "What more is required in order to launch the product?" if the physical product is available, rather than simply a set of descriptive artefacts.

## Replacement systems are large batches?

Incremental delivery clearly lends itself to the delivery of new systems. However, when a legacy system is to be replaced, the opportunity is not as obvious. After all, no value can be accrued until the existing system has been replaced. Or can it?

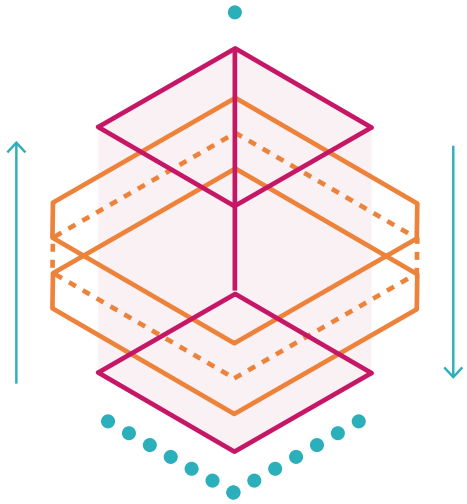
Unsurprisingly, the answer again lies in the business case for the replacement system, i.e. where will value come from? The reality is that systems are rarely replaced simply due to obsolescence, and the overriding factor tends to be either high running costs or a need to support new initiatives: better customer service, omni-channel integration or improved operational efficiencies. Often the reality is that the value of the opportunities far outweighs the potential cost savings, and therefore an incremental approach can be taken to realising their value.

Consider the not entirely hypothetical situation in which a police force needs to replace its aging crime scene reporting system – a legacy platform – which requires officers to record information at the scene using their trusty notepad (pen and pencil, not tech), and then transcribe the notes into the system at a later date; a hugely inefficient process and a waste of valuable time when the officers could be on the beat.

The 'natural' intuitive assumption is there is no value in the new system until the old one has been replaced, and therefore the new system has to cover all eventualities. The range of crime scenes being extensive, this leads to one huge batch. However, if we focus on realising value through process optimisation rather than focusing on the replacement system, the picture looks quite different.

The proposed new system will provide officers with tablets (tech not pharmaceutical) into which they can record information directly (including the time-saving options of photographs and audio recording). Even by the most conservative estimates, this will save officers over four hours per week, and thus, for a small police force such as Manchester, the saving is circa £18,000,000 per annum. But can a proportion of this be achieved incrementally?

Adhering to the approach discussed above, we can start by segmenting the value. Given that the aim is efficiency, the maximum incremental value is gained by focusing on the crime type on which most time is spent: burglary! The logical first step is to record the simplest form of burglary: burglary on a single premise with no witnesses and no additional complexity. Next, this can be extended to include witnesses and multiple premises. While the value at this stage is limited, the early increments provide the opportunity for feedback and learning. It is also possible to test key technical challenges such as security and integration with the legacy system (to facilitate parallel operations).



Having completed a sufficiency of functionality to support burglary reporting we can then move on to the next increments: car theft, domestic violence, fraud etc. Hopefully it is clear that once a small number of crime types are addressed, significant value can be released through parallel running of the old and new systems, and importantly long before we approach the crime types with low instances (e.g. the implementation of salmon handling or queue jumping) the system will address over 95% of all crimes committed, thus releasing the vast majority of the value long before the old system is retired.

It is not always possible to release a system early, but that does not mean that significant value cannot be gained from an incremental approach. The costs of parallel running may be prohibitively high, or the new business process may differ significantly from that which is being replaced requiring that the first major release be a large batch, but numerous options still exist.

Deploying incrementally to a model office facilitates testing, data migration, training, and business process reengineering all to be undertaken in parallel with development, significantly shortening time to value and reducing risk. And, of course, working in this way will help to identify the real MVP.

## Component-based vs incremental

The Student Loans Company is currently replacing its core loan system. The aim is to provide a digital customer-centric journey, which will make life easier for students and significantly reduce the call centre requirement. Broadly speaking, there are two logical approaches to this: to break the system into components, allowing teams to work in parallel before a final big-bang integration; or to deliver the system incrementally.

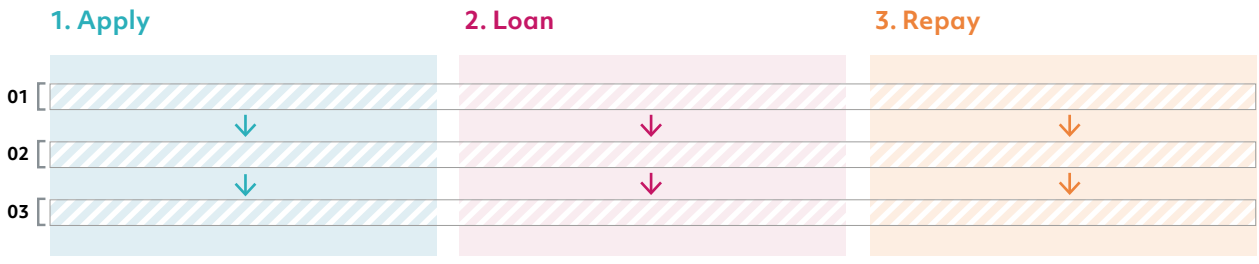
The component-based approach looks at the system in three parts: apply, loan, repay. Implementing each separately may look like "thinking small" but each is large and complex in isolation, giving three large batches followed by a "super-batch" big-bang integration.

Importantly, evidence of progress within the batches is limited as there are no value milestones.

Approaching the problem from an incremental perspective, it is quickly evident that the largest demographic group is also the simplest from an implementation perspective. UK-based students who qualify for a loan, go to a single university and then get a job and repay the loan, represent a significant population of the user base. Building for this group first proves the end-to-end system, and offers a clear value milestone for measurement. Adding complexity to this first increment to cover changes at university and employment quickly provides a set of functionality that addresses the majority of UK-based students.

### Component-based approach


### Incremental approach




**Key**

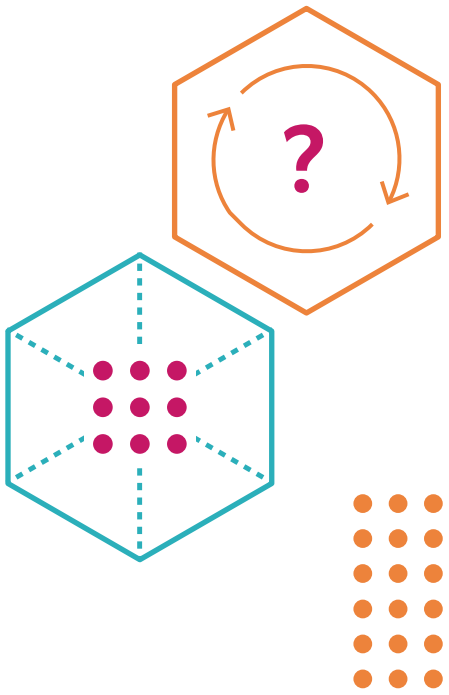
Increments

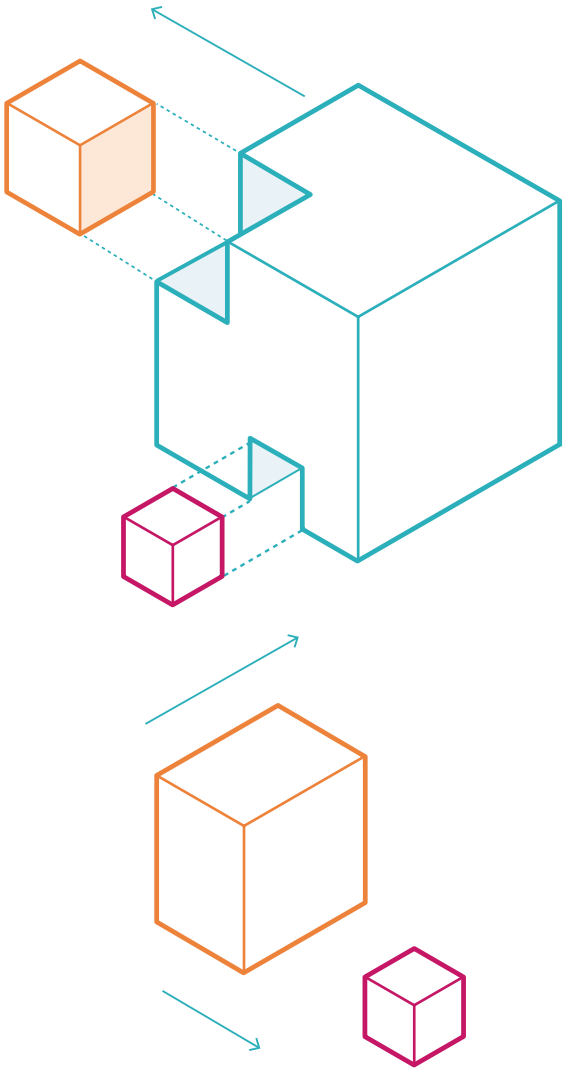
# Conclusion

Returning to our theme, Think Small to Deliver Big, hopefully it is now clear that even the largest problems can be broken down into smaller, simpler steps that reduce complexity and risk, and deliver early value.

Early in the process, it is important to address uncertainties: identify unknown unknowns and ensure the journey won't be derailed by gotchas. But this should not itself become a big-batch activity. Think about each area of uncertainty and address each in short iterative bursts with clear measurement points. Only undertake sufficient up-front activity to enable delivery to commence with a degree of confidence.

To maximise the value from an iterative, small-batch approach, deliver incrementally. Focus on where true lies: customers, functionality, service. Don't let the ultimate solution blind you, and enjoy the journey.





## Want to discuss how to think small to deliver big?

At Scott Logic, we design and build software that transforms the performance of some of the world's biggest and most complex organisations. By taking an incremental approach, we help our clients to reduce risk and deliver early and on-going value.

If you'd like to discuss how your organisation can adopt an incremental approach to designing and delivering its products and services, we're always happy to chat.

**To arrange a free consultation, contact Rob Smith on:**

+44 333 101 0020

sales@scottlogic.com

SCOTT LOGIC / ALTOGETHER SMARTER

3rd Floor, 1 St James' Gate
Newcastle upon Tyne
NE1 4AD

+44 333 101 0020

scottlogic.com