

## Why Your Business Needs Agile Software Development

*Flexing IT to meet your business needs*

Spring 2006



## Why Your Business *needs* Agile Software Development

Flexing IT to meet your business needs

### Executive Summary

Organisations must be agile and responsive to prosper in today's fast-changing business environment. They must swiftly adapt their products and processes to respond to their customers' changing needs. As software becomes increasingly integral to their efficiency and effectiveness, many organisations are discovering the biggest hurdle to their business agility is the development processes used by their IT departments.

In many organisations, the legacy of early software productivity initiatives remains entrenched. The Waterfall - or V - model, its processes and associated behaviours, continue to be widely employed in IT departments despite the barriers it raises to change and innovation, well-documented rates of project failure and widespread customer dissatisfaction.

*At least 45% of features and functions built into the average software product are never used*  
(see page 6)

Modern approaches, based around iterative and incremental delivery, are designed to efficiently adapt to change throughout a project's lifecycle. They are based on quality techniques proven in product development and deliver levels of flexibility and responsiveness previously unachievable in software development.

By adopting these Agile approaches, organisations can respond swiftly to customer needs, speed solution delivery and adapt efficiently to change.

---

### The Software Development Paradox

The software development industry is a paradox.

Software is so pervasive within our business and personal lives that we could legitimately ask ourselves if there has ever been an industry that has changed the world so much in such a short time.

Yet, our industry remains plagued by failure.

- Year after year, the much quoted Standish Chaos Survey reports that ***over two-thirds of IT projects fail to deliver*** 'on time and on budget, with all features and functions originally specified'<sup>1</sup>.
- The 2000 Chaos survey reported that ***23% of all projects were cancelled or never implemented.***
- Most businesses have a ***growing backlog*** of software development projects and consider their profitability to be ***constrained by IT's capacity to deliver.***

Despite this, the demand for new software far exceeds our capacity to reliably supply it. And ***industry is losing billions in revenue each year because of this.*** We are delivering value, but we could deliver much, much more.

*So what is wrong?*

## **A simple misunderstanding**

Looking back to the 1970's we find a young, innovating and immature software development industry staffed by talented craftsmen. This was an industry that was growing quickly and suffering for it, that had discovered the difficulties of scaling software development effort and that was rife with defects and rework. This was an industry that knew it had problems and was searching for solutions.

At about the same time, but in a very different industry and in a distant part of the world, the Japanese car industry was about to give their western competitors the shock of their lives by producing better quality cars, faster and more cheaply. And as the western car manufacturers started to learn about the Japanese quality techniques behind this paradigm shift, so too did the software developers.

The IT sages, thought leaders and industry advisors of the time looked down from their mountain of rework and decided they could borrow the quality techniques that were emerging from the manufacturing industry. And it is easy to see why. Manufacturing also had a high defect rate and a high cost of rework, so the Japanese had learned how to “build quality into the processes” rather than inspecting defects out. In particular, they borrowed the concept from Philip Crosby that quality is “conformance to specification”.

It was a good move for the western volume manufacturers and soon they were playing catch up with their eastern competitors. But it was a terrible misunderstanding for the nascent software industry. Although software development can learn much from both the Japanese quality and lean movements, quality techniques designed for manufacturing do not translate wholesale to software development.

***The quality model used by manufacturing simply does not apply to software development.***

### **Key learning point:**

Software development is a subset of product development, not a type of manufacturing. Product development is a learning activity, is iterative and involves repeated testing. It takes place before manufacturing, and its output is a detailed specification that is then used by manufacturing.

*So what happened next?*

## Enter the V-model lifecycle

Unfortunately, during the 1970's the quality movement was new to the west and the only information freely available was based on manufacturing. Manufacturing is repetitive with clearly defined outputs. Product development is iterative and explorative. Manufacturing starts with a correct specification. Product development ends with one.

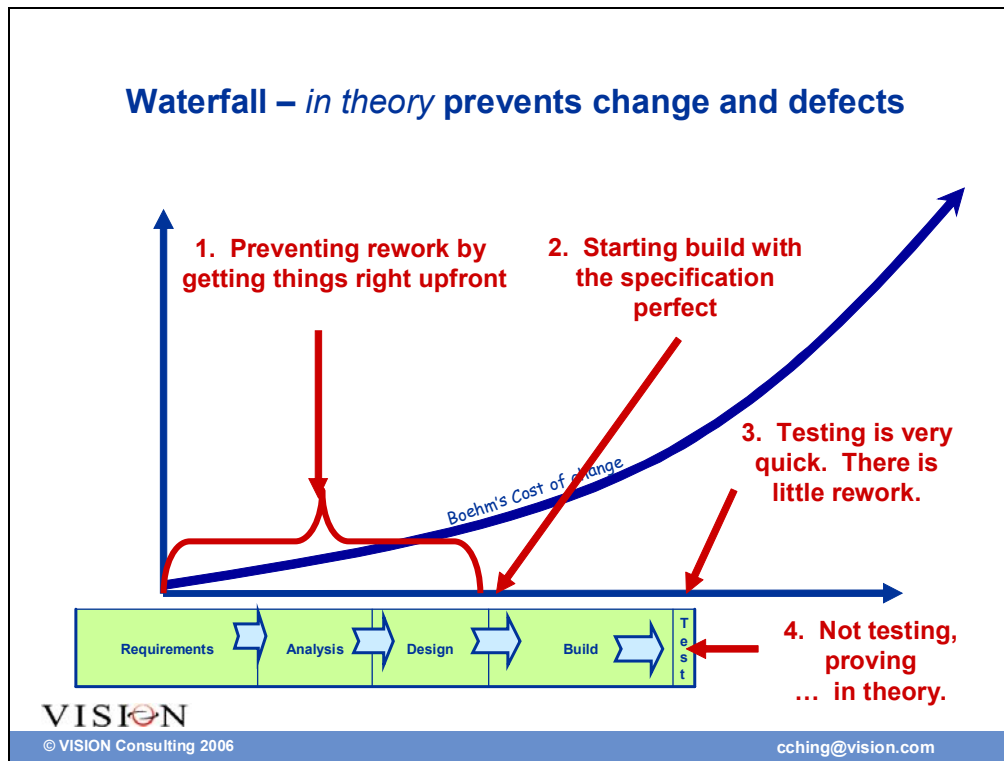


Figure 1 Waterfall in theory

So when the software development gurus borrowed quality approaches from manufacturing they came up with the Waterfall model (now more commonly known as the V-Model). Here the key to achieving high quality and low rework is to start with a clear and accurate specification, the theory being that if the specifications are right up front, expensive rework is prevented later.

Sadly, in practice, this lifecycle causes development teams to *struggle* to specify requirements fully and correctly up front, in the early stages of the project, before they have finished learning. The subsequent discovery of the true requirements as the project progresses then results in change requests. For instance,

- Respected software development researcher Capers Jones<sup>2</sup> reports that between **25% and 35% of requirements change** on large and very large projects; and

- Product development guru, Donald Reinertson<sup>3</sup>, reports that out of hundreds of projects he has studied there is ***no case in which requirements remained stable***.

So what does this mean for software projects?

## The V-model is devastatingly inefficient

Not only do requirements change, but the bad news is that ***the majority of change requests happen near the end of the testing phase when it is very expensive and time consuming to do the rework***. This turns what, in theory, is a very efficient development process into what is, in practice, a very inefficient, costly and time consuming use of expensive development resource.

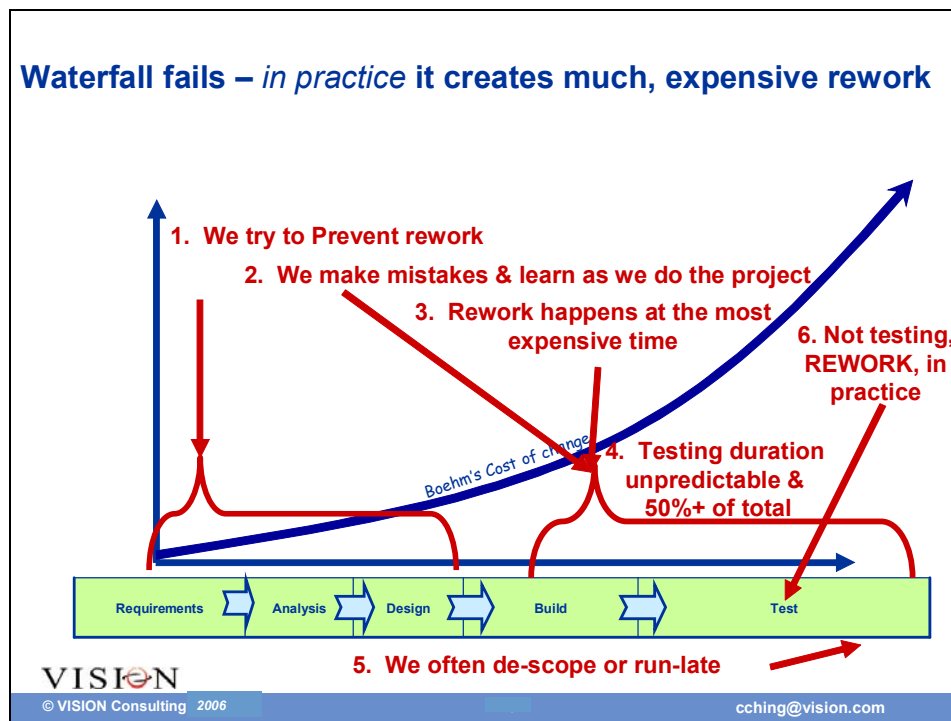


Figure 2

Given the high cost of rework, many change requests are therefore rejected in order to save work and protect the project's delivery dates. But given the high administrative effort involved in requesting a change, customers tend to request only changes that they consider important. This means that when their changes are rejected the ***customers, understandably, consider that the software has failed to deliver the functionality that they require***.

However, many changes ***will*** be approved late in the project lifecycle. Accordingly every

W Edwards Deming, the American statistician who taught modern quality techniques to the Japanese following WWII, taught managers to not "rely on mass inspection" at the end of a process because it was extremely costly, unpredictable and ineffective.

The V-model relies on inspecting defects out, just as Deming warned against.

Instead, Deming taught, managers to "build quality into their process".

experienced project manager builds contingency into their plans to absorb the rework and protect their delivery date.

But how much contingency should they include? If they plan for a short testing phase then they risk running late. If they plan for a long testing phase then it appears either as if they are padding or they are admitting that they expect testing to take a long time because their quality practices are inadequate. Neither solution is ideal.

Managers often plan for challenging, but optimistically brief, testing phases. An unfortunate and far too common result of this is that when the project reaches the promised delivery date, there is a tall stack of well-crafted documentation and a large suite of delivered, but not yet fully tested, software. Software which is essentially unusable until proven.

On realising the project will overrun because testing will take longer than planned, the customer and project manager must choose between:

- ✗ delivering potentially low quality product;
- ✗ de-scoping the product; or
- ✗ missing the delivery date.

None of these options impress the customer. Worse, the customer is often surprised when presented with them because the project had previously been reported as running to schedule.

*So what does this all mean?*

## ***Our Customers do not Trust us***

Trust and commitment are essential to success, yet our consistent failure to meet our commitments has the unfortunate side-effect that our customers don't trust us. And this lack of trust leads us to working in contractual, rather than collaborative relationships.

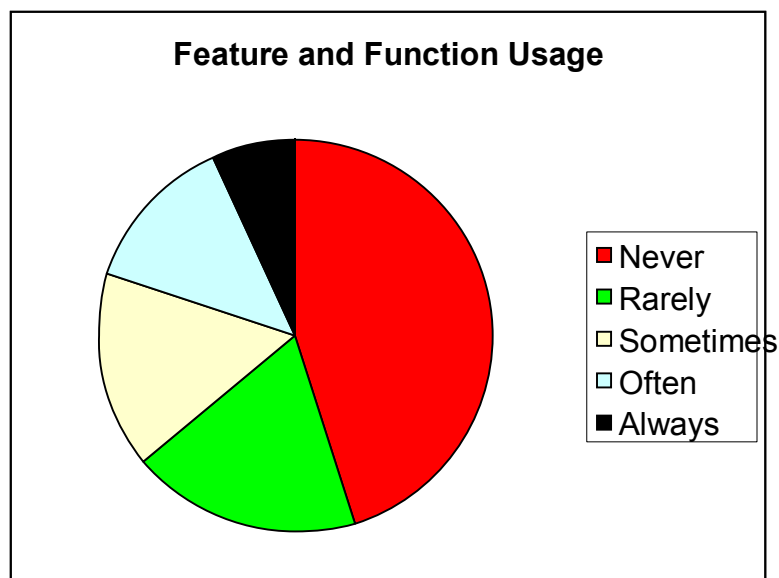
***This mistrust adds layers of co-ordination waste which slow down our development process even further.***

Consider the requirements sign-off process. When not working collaboratively this can become a contractual game of blame avoidance where anyone who might be affected by the requirements must sign-off the specification before design work can start. In larger organisations this can involve dozens of signatories and add dramatically to the delivery schedule.

A further problem is that when the requirements of different signatories conflict, this conflict is often resolved by making the specification vague enough that conflict disappears. A simple fix, which means these ambiguous requirements will result in extensive rework later in the project.

Consider too, projects that start with a schedule commitment which the project's development team consider impossible to meet. In these cases, the development team feels minimum ownership for the project and anticipates failure. Respected author, Ed Yourdon describes these projects as "Death March Projects". He details several reasons why such projects commence – including politics, optimism and naivety - but beneath all these reasons sits a low-trust culture where the negotiation between customer and the delivery team are neither open nor honest, and where it is unsafe to admit the possibility of failure.

One of the most remarkable side-effects of this contractual approach is that the software eventually delivered ends up bloated with unused features. One study, by Jim Johnson of Standish, discovered that 45% of features in the software studied were never used, while another 19% were rarely used (see Figure 3). Naturally, this unnecessary increase in the number of features has a corresponding effect in the complexity, size and cost of the software build.



**Figure 3 Bloated Software Applications.**

In summary, years of project failure, customer frustration and rising costs have shown that the Waterfall approach is an inappropriate model for software and product development. It is unreliable, creates distrusting relationships and inferior, yet bloated products. And as we shall see below, Waterfall is costing your business significant revenues.

*So what can we do about it?*

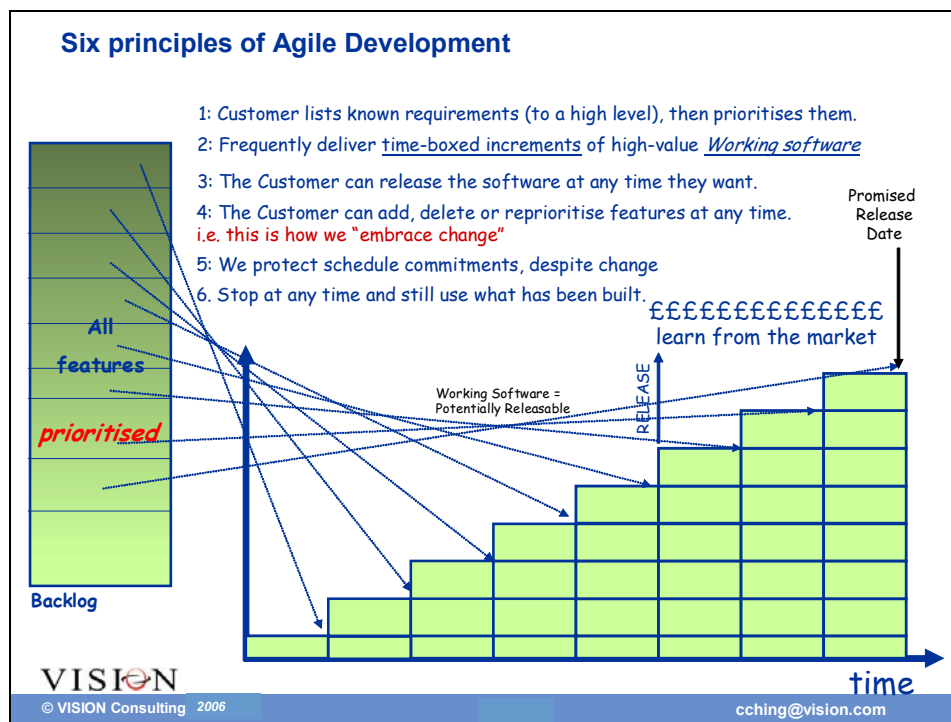
## Introducing Agile

Agile software development recognises that software development is inherently a type of product development and therefore a learning process. It is iterative, explorative and designed to facilitate learning as quickly and efficiently as possible. Quality, in product development, is judged according to the product's "fitness for use". This means Agile is designed to support the needs of today's flexible, responsive and adaptive organisations.

*So how does it work?*

## ***Iterative development***

Typically, customers are first asked to provide a prioritised list of features, but not to detail them in a full-blown requirements specification. Second, working closely with their customers to draw out the detail of the features, IT delivers small increments of well-engineered<sup>4</sup>, working software every few weeks. The customer and IT agree each iteration's features according to the customer's prioritisation and IT's risk analysis. Third, at any stage the customer may ask the IT department to release the working software product so that they can start using it. Fourth, the customer may change their prioritised requirements list while IT continues to deliver increments of working software from the top of the list.



This iterative approach enables customers to learn quickly because they use the product as it evolves and can incorporate their learning into the product. Further, the ability to release working software many times throughout a project allows the customers to learn from those actually using the product.

## Key Learning point.

The concept key to the high productivity of iterative development is “working software”.

Working software is fully tested software that is ready for release. It is not software that is nearly working, is yet to be fully tested and still requires rework.

This is a vital definition because it encourages a mindset of “building quality in” rather than “testing defects out” – the revolutionary approach Deming taught the Japanese.

Delivering “working software” at the end of each iteration has powerful benefits:

- productivity and reliability are improved because defects are found and fixed far sooner;
- software can be released to the users at the end of any iteration; and
- predictability is improved massively because schedule variation is contained within each iteration. Development teams can therefore confidently make then keep their schedule commitments.

*So that's it then?*

## ***No Silver Bullet***

While the iterative mechanism is conceptually simple, in practice it is demanding work. Successfully adopting Agile will require overcoming many obstacles and preconceptions.

- While there are many advantages in releasing software to the customers more frequently, many organisations are not used to working this way. Initially they find it challenging to adapt the processes of the areas involved – including Release Management, Training, Operational Risk, Capacity Management, end-users and even marketing.
- Working iteratively tends to highlight existing dysfunctionality and bottlenecks within the organisation that were previously hidden. For instance, if your DBA team currently has a 4-week service-level-agreement then it is impossible to deliver working software every two weeks without

The two best known Agile approaches are Scrum and Extreme Programming (XP):

- **Scrum** is a light touch method for planning and executing *iterative* development projects. It wraps around a team's existing engineering practices.
- **XP** also builds software iteratively and shares many project management practices with Scrum. Despite its unconventional sounding name, XP's power comes from its use of powerful quality and engineering practices.

tackling this bottleneck.

- Scaling Agile beyond an initial pilot project staffed by enthusiasts requires considerable change management skill.

Key to overcoming these obstacles is executive commitment to Agile adoption, an understanding of why Agile works across all levels and recognition that Agile didn't create those obstacles. Agile simply highlights obstacles that already exist and shows where to focus process improvement efforts.

*What does this mean in practice?*

## ***Risk Management***

Experienced project managers understand the importance of actively and formally managing project risk. They also understand that some problems will only be found by executing and proving the software, so they conduct "proof of concept" tests in high risk areas. In this way they manage risk based on fact and surface problems early when they are easier and cheaper to resolve.

The iterative mechanism intentionally uncovers problems early by forcing development teams to execute working software from end-to-end from the first iteration. Successful Agile teams make formal risk assessments throughout each project and deliberately move high-risk / high value features into the earliest iterations. For example, they may run what they call "spikes" which are quick, focused proof of concept tests to demonstrate viability.

## ***Productivity through Quality***

One of the most noticeable things about developers working on successful Agile projects is the tendency to test as early, as often, and as efficiently as they can. They consider it key to their high productivity.

Preventing defects by *writing tests before writing the code* tackles the enduring problem of ambiguous requirements. The thought that goes into writing these tests is a powerful lever for teasing out the ambiguity from requirements. And automating these tests provides a safety net for developers, giving an early warning if they inadvertently break the tests.

## ***Building Trust***

Interesting things happen when IT starts making commitments it can keep.

By delivering working software in fixed time-boxes, Agile projects ***reliably and repeatedly deliver the product most fit-for-purpose*** to the customer ***on-time***.

Agile makes it ***easier for the customer to plan and execute*** their own programme of work. The increased predictability in their working lives ***makes it easier for the customer to make and keep their own commitments and promises***. So, not only is the relationship between IT and Customer improved, but also the relationship between the customer and their customers.

***As trust builds over time, many of the hidden co-ordination wastes become obvious and can be easily eliminated.***

For instance, when the development function reliably delivers on time and provides credible visibility of the project's true progress throughout its execution, both the customer and the project management team can be confident that they are in control. As predictability, visibility and trust increase, game-playing diminishes.

### Key Learning point.

One of the key changes instigated by successful Agile projects is the realignment of the responsibilities and accountabilities of the customer and IT teams. Agile provides customers with an efficient engine and puts them firmly in the driving seat. If they navigate poorly or drive with the handbrake on then they can't blame the engine for low fuel economy or for ending up at the wrong destination.

Successful Agile projects negotiate this risk by clearly agreeing the Customer's responsibilities with them and ensuring that the customer works closely with the development team, provides timely feedback and actively manages their prioritised feature list.

This change is about more than just clarifying responsibilities.

It is about working more collaboratively.

It is about IT being seen as a key enabler of product and process innovation, not an obstacle.

It is about IT becoming an essential partner in value creation.

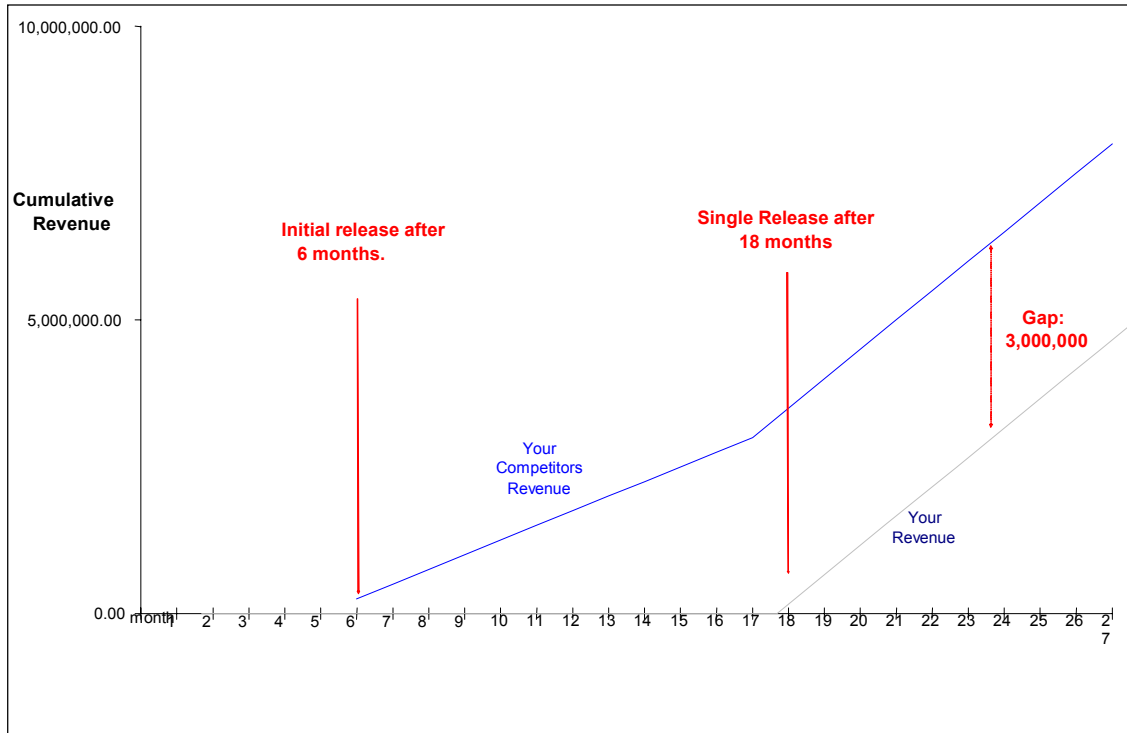
### ***Developing Products Faster***

With the Agile approach, IT has the opportunity to ***significantly influence the profitability of each project.***

Taking a simple example:

- Your organisation sees a gap in the market and sets up a product development project to deliver a new product in 18 months time. It is forecast to earn £6,000,000 each year once it goes live.
- However, your competitor sees the same gap and initiates an Agile Development project to create their product.
- After 6 months - on the same day that you get final sign-off for your requirements documentation, they have built the most valuable 25% of the product.
- Because they've built the product in an Agile way, they launch the product into the market where they start earning £250,000 in revenue each month.

**In this conservative example, assuming your project finished on time, at the end of 18 months your Agile competitor has earned at least £3,000,000 and you have earned nothing.**



Further, their product will be superior because they have feedback from the customers - those already purchasing and using the product - and are building that experience back into the product.

## Generating value

This increased speed to market means IT has the opportunity to accelerate revenue generation and increase profitability because Agile projects are much **faster to market** than traditional projects.

This speed comes from a number of areas.

1. The software is typically smaller so the project takes less time. Since the project is delivered in priority order, most of the value from the project is delivered early on. Project sponsors can choose to divert their resources to other projects before building lower priority features (especially the 45% of features that may be redundant).
2. Unlike waterfall projects where every approved change request results in expensive rework, many changes in an Agile project are free or have a very low cost. Often a change is as simple as an update in a spreadsheet or requirements document.

3. The average cost of change is much lower because developers strive to be working from well-designed and defect-free software. Strong engineering practices are vital to achieving this.

Finally, since Agile development approaches are highly efficient, Agile development teams **do more projects and reduce the organisation's backlog of outstanding projects.**

*So how do we know it works?*

## Agile in action

Iterative development has a long history of success, for example the primary avionics software on the first space shuttle was developed using iterative development techniques. And perhaps the most significant endorsement of iterative development comes from the world's largest purchaser of software, the US Department of Defence<sup>5</sup>. In the 1980's, DoD issued standard DOD-STD-2167 which established policy for all software development and procurement. It was based on the Waterfall method. By the late 90s, however, due to the growing body of research highlighting the failures of the waterfall approach, it was superseded by MILSTD-498, a new standard which required software projects to be developed using iterative and evolutionary techniques.

Early adopters of Agile have included ecommerce giants such as Amazon, Yahoo and Google, while it has now crossed into the mainstream into larger and long-established organisations. In the UK, BNP Paribas<sup>6</sup>, Egg Internet Bank<sup>7</sup>, Standard Life<sup>8</sup> and British Telecom<sup>9</sup> have all declared success using Agile approaches.

The growing body of published research increasingly shows the benefits of Agile and iterative development:

- **Productivity is increased.**  
A 2002 study<sup>10</sup> of over 40,000 projects reported that developers working on projects using rigorous iterative or evolutionary development techniques delivered 570 function points per full-time equivalent developer, compared with only 480 function points for developers working on rigorous waterfall projects.
- **Staff Morale is improved**  
A study of teams using Extreme Programming at Motorola<sup>11</sup> showed that overall, 85 percent of the developers enjoyed using XP, 68 percent stated that using XP increased their job enjoyment, and 79 percent stated that, given a choice, they would definitely use XP again.
- **Product quality is improved**  
Primavera<sup>12</sup>, a vendor of enterprise project management systems, reported a 30% improvement, with the number of customer-reported defects in the first nine months dropping from 0.51 to 0.36 defects per KLOC.

## In conclusion

In today's fast-changing business environment, organisations must be flexible, responsive and adaptive. They must deliver new products to market promptly, create products that meet their customers' needs and adapt these swiftly as those needs change. And they must do all this cost effectively.

Yet many organisations still find their biggest hurdle to being flexible and adaptive is their approach to software development.

Agile software development approaches have been successfully adopted by large organisations in many industries to both increase productivity and increase revenue. Their adoption can be demanding but they do deliver breakthrough benefits.

Agile Software development will allow you to:

- produce products that are *fit for purpose*;
- deliver to significantly *shorter* and more *reliable* and timescales;
- *reduce your costs* by delivering more projects, more efficiently, often by fewer staff; and
- significantly *increase* your profitability and return on investment for each project.

Agile development will return *value* to your organisation and *benefit* your business.

**Table 1. The differences between Waterfall and Agile projects**

	Waterfall / V-Model	Agile
<b>Product</b>	An often bloated product that is still missing features (i.e. rejected change requests or features de-scoped to meet deadlines)	<b>The best product according to customers own prioritisation, incorporating learning from actual use.</b>
<b>Schedule</b>	Deadlines are often missed and it is highly unusual for a project to deliver early.	<b>Very high probability of meeting fixed date commitments. Can often deliver early, having delivered the highest value, thereby allowing funding to be diverted to other more profitable projects.</b>
<b>Quality</b>	Defects must be tested-out very expensively (and unreliably)	<b>Quality is built in. Quality is the key to productivity.</b>
<b>ROI</b>	Revenue stream initiation and value creation are delayed until the lowest priority features are delivered.	<b>Value is generated early, as soon as the minimum key feature set is delivered. Smaller projects mean lower cost per project. More projects are delivered. Greater return on investment.</b>
<b>Relationships</b>	Contractual.	<b>Collaborative.</b>