resillion

Quality: Engineered. Tested. Assured.

Navigating agility in the software development lifecycle:

a strategic perspective

Introduction

Agile principles enable organisations to adapt swiftly to change. However, taking an agile approach to delivering software is often not straightforward. Many organisations still struggle to align software development team objectives with the strategic goals of the business.

Relying solely on cost, quality and time as the primary drivers for decision-making in the software delivery lifecycle (SDLC) is restrictive. Shifting towards an agile mindset requires a broader perspective, effectively addressing scope, business value and risk. By taking this approach, time and quality become intertwined, and a risk-adjusted return on investment becomes crucial.

Although we've tried and tested a standard approach, we also acknowledge several important variables:

- B2B products need a different approach than B2C
- a company with 12 employees will require a completely different approach to one with 500
- pure software deliveries require a different approach than software combined with (embedded) hardware deliverables
- in-house development needs different solutions to outsourced (or hybrid) software development.

You'll see that we don't mention specific tools or suggest any particular technologies. This is because an agile mindset and approach should be achievable with any up-to-date tools and technology stack.

In future papers we'll elaborate further on this mindset and approach. We'll look in detail at the engineering and organisational choices that enable you to get fast incremental and qualitative software delivery and consider quality, cost and agility from a quality assurance (QA) and quality control perspective.

To truly embrace agility, the underlying ethos and processes must be woven into the strategy of the organisation.

Strategic perspective

In an attempt to identify what sets a high-performing organisation apart from low-performing ones when it comes to the software delivery lifecycle, the well-known book 'Accelerate'[1] introduced four key metrics that measure the productivity of software delivery teams, categorised into two main attributes: Throughput (or Tempo) and Stability.

Throughput

1. **Lead time for changes:** Time taken to deliver code for a feature (from code commit to deployment for end users).
2. **Deploy frequency:** Number of software deployments in production, typically reported per day, week or month.

Stability

3. **Change failure rate:** Ratio of failed changes in production to all changes.
4. **Mean Time to Recovery (MTTR):** Time taken to restore after a failure in production.

These **four key metrics** are foundational elements of the annual State of DevOps report[2], originally published by the DevOps Research and Assessment (DORA) institute and since 2018, published by Google Cloud. This report identifies elite, high, medium and low performing teams by industry sector and seeks to provide a reference point to help organisations continuously improve their individual performance. The State of DevOps report can be accessed via a survey and is available so that anyone can quickly check their own performance[3].

Does this mean that to become a high performing organisation a DevOps engineering model is essential? Not necessarily. Metrics are always useful, but it's important to understand their limitations in this instance. Here, they are only measuring productivity, not how effectively business value is generated.

An organisation can be a high performer without adopting DevOps, although they are likely to be following the logic and reasoning that sits behind delivery methodologies like Scrum or DevOps.

In the book 'Modern Software Engineering'[4] these guiding principles are presented independent of any software delivery methodology and support the idea that high-performance need not be linked to specific technologies or tools.

The most important takeaway from the 'Accelerate' metrics is that there is a correlation between throughput (or speed) and stability (or quality). The route to speed is high-quality software and the route to high-quality software is speed of feedback.

[1] N. Forsgren, J. Humble and G. Kim, Accelerate: Building and Scaling High Performing Technology Organizations, IT Revolution, 2018.

[2] Google, "State of DevOps Report," https://cloud.google.com/devops/state-ofdevops/

[3] Google, "DORA | DevOps Quick Check," https://dora.dev/quickcheck/

[4] D. Farley, Modern Software Engineering: Doing What Works To Build Better Software Faster, Pearson Education, 2022.

Cost and risk

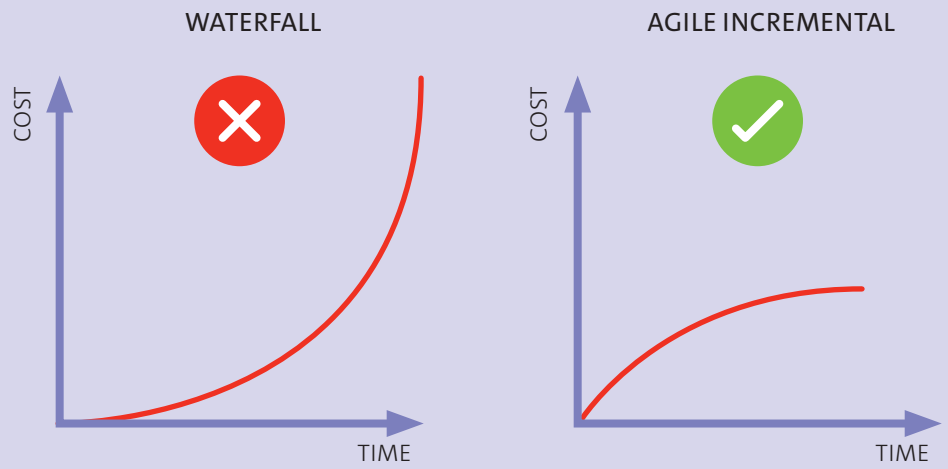

Figure 1 Classical waterfall model vs an agile incremental model that has throughput and stability with an 'inspect and adapt' mindset. A transformation to throughput and stability together with an 'inspect and adapt' culture leads to a stable cost over time, compared to an increasing cost over time with a waterfall software delivery model'[4].

Adopting the mindset that throughput and stability belong together and embracing an organisational culture of 'inspect and adapt' can lead to a more predictable and less risky cost model. Inspection and adaptation are foundational principles in The Scrum Guide[5].

[4]D. Farley, Modern Software Engineering: Doing What Works To Build Better Software Faster, Pearson Education, 2022.
[5]K. Schwaber and J. Sutherland, The Scrum Guide (edition November 2020), 2020.

Classical waterfall: cost of change

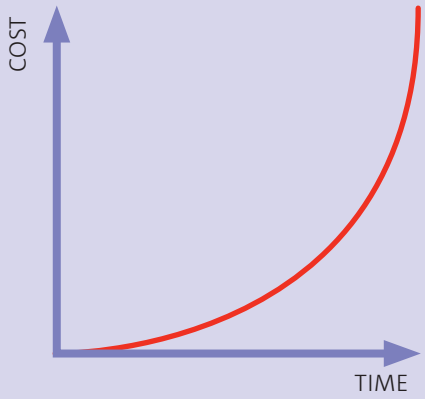

Figure 2 Classical waterfall cost[4]

With classical waterfall thinking, the most important decisions are taken at the beginning of a project and change becomes more expensive as time goes on. At the beginning of a project, knowledge levels are at their lowest and decisions are often based on educated guesses instead of accurate data. Not every piece of the puzzle is fully understood at the start and changes are not yet planned in.

Agile high-throughput and stability: cost of change

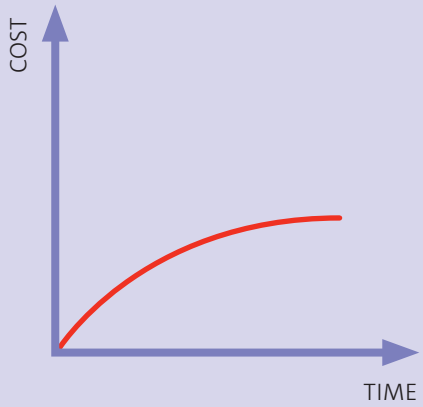

Figure 3 Agile high throughput and high stability software delivery cost[4]

With an agile approach, the cost of change becomes flatter over time. You'll reduce the amount of time spent on analysis and design, where no business value is accessible to the end user. Upfront activities are compressed into iterations and feedback from end users (either directly, or by a business representative) can then be taken into account in subsequent iterations.

Wherever software delivery teams can work independently, you'll see a flattened cost model alongside an 'inspect and adapt' mindset. Investment in rapid feedback with automated quality gates is needed to achieve high throughput and stability.

However, in situations where there is a strong inter-dependency between software delivery teams, any speed or quality issues can quickly become a bottleneck for the business. Proper technical and architectural alignment needs to be addressed before you can move towards more independent working models.

Change can come from many sources: improved ideas, learning from fixing defects or from customer feedback. The cost of change is under control in this model because iterations can be viewed as part of a defensive design strategy.

Each iteration is another opportunity to learn, react and adapt as your knowledge increases. If the value of a feedback cycle is negative, there is an opportunity to improve, without cost being out of control. If there is positive feedback, then there is an immediate return on investment and cost will be reduced.

Risk-adjusted return on investment

High-throughput and stability combined with an 'inspect and adapt' mindset (from now on referred to loosely as 'continuous delivery') flattens cost because as a defensive design strategy, it leads to risk-adjusted return on investment.

As an example, if you have 1M EUR to spend on a ten-month classical waterfall project, for the sake of simplicity, this means spending approximately 100K EUR each month. If the right decisions are made at the beginning of the project, you start seeing a return on your investment after ten months. If you make any poor decisions, your cost will go up and any return on investment will be delayed. In the worst case, when the project fails to deliver any of the expected business outcomes, you'll see a complete loss of the 1M EUR.

If you were to spend the same 1M EUR using a continuous delivery approach, after two months a minimal viable product would make it to production stage and you're already seeing some return on investment. You've also got the additional benefit of being able to introduce improvements based on actual (empirical) customer feedback. This is risk-adjusted return on investment in action. In this case, if the expected business outcome is not as good as anticipated, the project can be stopped early without spending the whole of the 1M EUR.[6]

Companies following an adequate 'inspect and adapt' mantra, embrace changes and allow for experimentation instead of resisting changes.

[4]D. Farley, Modern Software Engineering: Doing What Works To Build Better Software Faster, Pearson Education, 2022.
[6]D. North, "Scaling Agile Delivery Turing the lights on - Agile tour Vienna 2015," 2015. https://www.youtube.com/watch?v=mWXTYNhz-sk

Transformation and agile scaling

Getting the cost of change under control is relatively straightforward if you've got a greenfield project, or you're a start-up company. The challenge is how to implement a continuous delivery model in a brownfield situation.

Many companies set about it in the wrong way and stick to 'agile' as if it were a religion. They go to church, but don't actually believe in the set of values and ideals which lie at the heart of it. They are agile by name only, which can result in even worse misalignment between business and engineering than if they had stuck to a waterfall way of working. Bad agile implementation can be a worse enemy than using a slower process that's already known and proven. In this situation, bringing the whole business up to speed with agile values and ideals is the first step. Bringing in external guidance to help with self-reflection can also speed up the transition to agility.

When a company uses terms or metrics such as 'team velocity' or 'lines of code', it can be a sign that agility is superficial. These measures don't cover productivity or business value and are possibly harmful for assessing team productivity.

In situations where software delivery teams are not allowed to experiment, manage their own work, or acquire the right competencies, their motivation can dip dramatically. The teams feel little sense of mastery, autonomy and purpose. Instead, they feel emotional and economic pressure to succeed. They stop adapting, learning and putting the best effort into their work[7].

Organisations also incorrectly assume that multiple teams delivering software in the same iterative cadence represents agile maturity.

Agile maturity is improving value delivery iteratively to the end consumer

An agile cadence is of no use if the value delivery (or business impact) to the customer slows down. 'Inspect and adapt' is a mindset, a culture and the process follows on from there. This is not the same as not having a plan.

Delivering customer value is the main priority and allowing experimentation and learning to get there more efficiently is the goal. Eliminating low value improvements should also be included.

In a brownfield situation, adopting agile methodologies such as Scrum, Kanban or XP is very hard. These methodologies which are typically very successful when used by smaller start-ups, scale badly for larger brownfield projects.

SAFe (Scaled Agile Framework[8]) for example, is an attempt at scaling agility. But with a single hardening or learning iteration at the end, it doesn't offer a stable cost model strategy, or a culture of continuous improvement.

Undertaking the transformation to a continuous delivery agile model is challenging but according to Accelerate, it will be rewarding[9]. There's no one-size-fits-all and each organisation must overcome the hurdles one by one. The DevOps Handbook comes with excellent suggestions on how to start your cultural transformation. We'll also cover more on this in our upcoming paper on the engineering perspectives of the software development lifecycle.

[7]C. M. Rebelo, "Agile's worst enemy is not waterfall - is bad agile," 14 12 2018. Available: https://www.linkedin.com/pulse/agiles-worst-enemywaterfall-bad-agile-cristina-moura-rebelo.
[8]"SAFe - Scaled Agile Framework," https://scaledagileframework.com
[9]Gene Kim, Jez Humble, Patrick Debois & John Willis, The DevOps Handbook, IT Revolution Press, 2016

Let's now look at why a loosely coupled architecture is good practice. From an engineering perspective, if system components are tightly coupled, each component depends directly on others. A tightly coupled design can impede an 'inspect and adapt' mindset because change in one area ripples through to other areas. Any component can quickly become a bottleneck for high-throughput and high-stability of the system.

Tight coupling strongly limits the ability of an organisation to transform to agile or to scale up. Loose coupling of organisational components and independent work streams is good practice.

Tight coupling also requires extra communication, additional alignment on work, more rigorous processes and a central decision-making authority to deliver business value. Any misalignment can become a bottleneck for swift and adequate business delivery. (Figure 4)

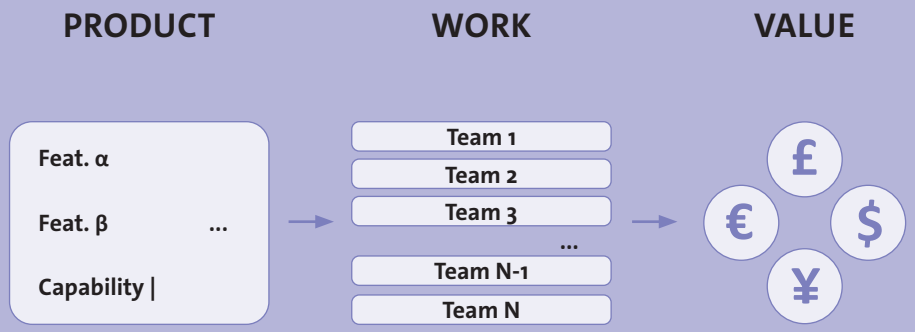

Figure 4 Misaligned scaling with tight coupling.

On the flip side, loosely coupled organisations don't need any direct involvement with other areas. Any misalignment would not impede business value delivery, because alignment is not essential. Risk-adjusted return on investment also applies when adopting loosely coupled design to your organisation. (Figure 5)

Loose coupling of teams and technologies enables agility, resilience and with that greater quality. Furthermore, increasing headcount is easier to manage because enabling new, small work stream-oriented teams is simpler.

Here are some of the signs of a poorly scaled and misaligned way of working:

- throughput decreases (or worst case, comes to a halt) if resources are added to your project in its current way-of-working
- development teams spend more time in meetings instead of delivering potentially shippable code
- production systems become unstable when deploying more frequently or when adding resources.

Keep in mind that any agile methodology is only a piece of a much bigger puzzle, one that can only be solved using a change of culture and mindset.

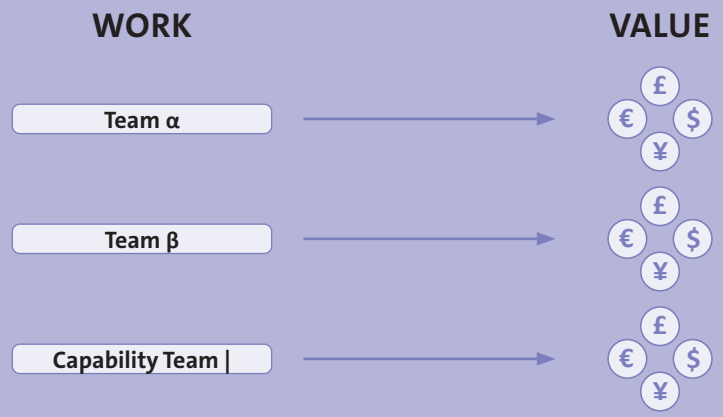

Figure 5 Loosely coupled aligned agile scaling. Minimise input, maximise output.

Conclusion

In conclusion, embracing agile principles requires a strategic alignment of software development with broader business objectives. At Resillion, we understand the challenges and nuances of modern software delivery. Our 700+ experts worldwide offer end-to-end digital testing services, ensuring your products are high-quality, secure and aligned with your strategic goals.

Start your quality transformation

Find out more about our Quality Engineering services on our website or get in touch with us today.

resill!on

Quality: Engineered. Tested. Assured.