DevOps, Platform Engineering, and Newtonian Operations: Rethinking change and stability in delivery

By Mike Peachey
Principal Platform Architect

There is a tendency in the industry to view a project that is not being changed as "at rest", and a project that is being changed as "in motion", but this has led to wasteful investment in change that serves little purpose.

- A project not being changed is not at rest, it is in motion at a constant velocity
- A project being changed is not in motion, it is changing velocity (Δv)

While "velocity" has been appropriated by agile frameworks, we are using it here in the Newtonian sense, as a vector quantity with both magnitude and direction.

Applying Newtonian thinking to project delivery could be transformative, addressing change as the means to invent or improve, but normalising the use of necessary and sufficient change only when appropriate, not simply as a demonstration of agility.

Compounding this approach with the use of accelerators that preconfigure a project or a solution, we can accelerate a project directly to a functioning velocity with very little cost, minimising the need for changes and maximising business value.

Lex I: Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus a viribus impressis cogitur statum illum mutare.

Law I: Every body perseveres in its state of being at rest or of moving uniformly straight forward, except insofar as it is compelled to change its state by the forces impressed.

Sir Isaac Newton, Philosophiæ Naturalis Principia Mathematica, 1687

Challenge 1: Newtonian Operations

The effort to establish the Platform Engineering role has eclipsed the criticality of the Operations role.

What is meant here, is that just because a project is not undergoing active modernisation, migration or feature development does not mean it is stagnant. It lives and breathes and requires Operations role(s) to keep it alive. Operations acts to counter resistant forces that would deplete the momentum put into place by Development and Platform Engineering. Without operations, the project will decay until it must be replaced. With good operations, the limit of a project's lifespan is only its usefulness to the business.

Operations and Platform Engineering are not the same

Platform Engineering is closer to Development than it is to Operations. If it wouldn't cause confusion, it would be better to call them "Platform Development" and "Application Development", or "Platform Engineering" and "Software Engineering", but the terms we have are the terms we must work with to achieve common understanding.

Platform Engineering is about creating the platform that the software applications run on, and the software development lifecycle tooling that the software is developed with. Operations is about keeping the platform running, and keeping the software and its data secure, durable, available and under observation.

Platform Engineering and Software Development however are much more similar. While being very distinct from one another, they share a creative nature:

- Developing software requires highly-specialised depth-first skillsets and generally produces flexible and mobile artifacts
- Developing platforms requires highly-specialised breadth-first skillsets and generally produces rigid and immobile artifacts

Operations picks up from both disciplines, acting as a customer and supporter of both, focusing on robust and secure delivery and support of the production system and its consumers. Too often the question is asked **"Who will be responsible for managing this service once it has been delivered?"**, and no answer is yet known.

It is a lot more obvious for a system that will be used by the public, but in other cases can be overlooked. Whether a presumption that Development will continue to manage the service, or that no management will be necessary, or that it is a problem for the future, the result is the same: inefficiency, instability, misallocation of people and resources, missed opportunities, and decay.

What is the solution?

ideally the chronology of a project might be:

1	The Platform role creates the core platform, and the software development lifecycle tooling.
2	Platform hands over the core platform to the Operations role.
3	Platform hands over the software development lifecycle tooling to the Development role.
4	Platform continues to develop any bespoke platform requirements that are non-blocking.
5	Development creates the software.
6	Development, Platform and Operations work together to deliver the software to production.
7	Development continues to develop the software as required by the business.
8	Operations keeps the platform running, and the software running on the platform.

Do not take this as a suggestion that the people working in these roles must be separate, or that there must be independent teams. As highly skilled individuals with wide and varying levels of experience, sometimes the roles can be performed by the same people, or by the same team, but the roles and management of their tasks must be understood to be distinct.

To assume that any skilled technical person can properly fulfil all of both or all three roles is short-sighted, but filling roles with the right portion of time from the right people at the right time can be perfectly astute, so long as you don't lose sight of how any compromises may be addressed in the future.

The larger a project or a business, the more necessary it is to have distinct teams per role, but the smaller a project or a business, the more likely it is that the roles will be performed within a single team. This likely will be at the cost of a lack of depth of experience in each role, and a much higher dependency on reusable patterns and accelerators to achieve high-quality, simple and low-maintenance results. It is in these smaller projects, where employing a high quantity of individuals with deep but narrow experience is not feasible, that the most benefit can be gained from the use of accelerators.

The other key risk of misaligning these roles and tasks is in project management. So many projects have suffered from the failure to properly handle operations tasks when there is no dedicated Operations team. "Maintenance", "patching", "support activities", etc. are all too often seen as not delivering value and are left to the end of the project or are not done at all, and this can be a costly mistake.

Operations tasks aren't "not delivering value", they are "maintaining value". They are the tasks that keep the project running and delivering value to the business. They are the tasks that keep the project from decaying until it must be replaced. However, often they do not align to the "agile" mindset. They are not "valued features", they are "maintenance costs".

- How do you insert repetitive tasks into a sprint?
- How do you justify the delivery time for those tasks to stakeholders?
- How do you prioritise them over features?
- How do you measure the value of them?
- How do you measure the cost of not doing them?

Some projects will use a dedicated “maintenance” epic that never ends, to which a minimum number of points per sprint must be allocated. Some may reject the idea of an epic that isn’t finite. Projects that use Kanban are more adaptable to operations tasks. Some will create an independent board for operations tasks that reports velocity independently of the main development work. There are many approaches, but those that attempt to shoehorn operations tasks into a development workflow often suffer for it.

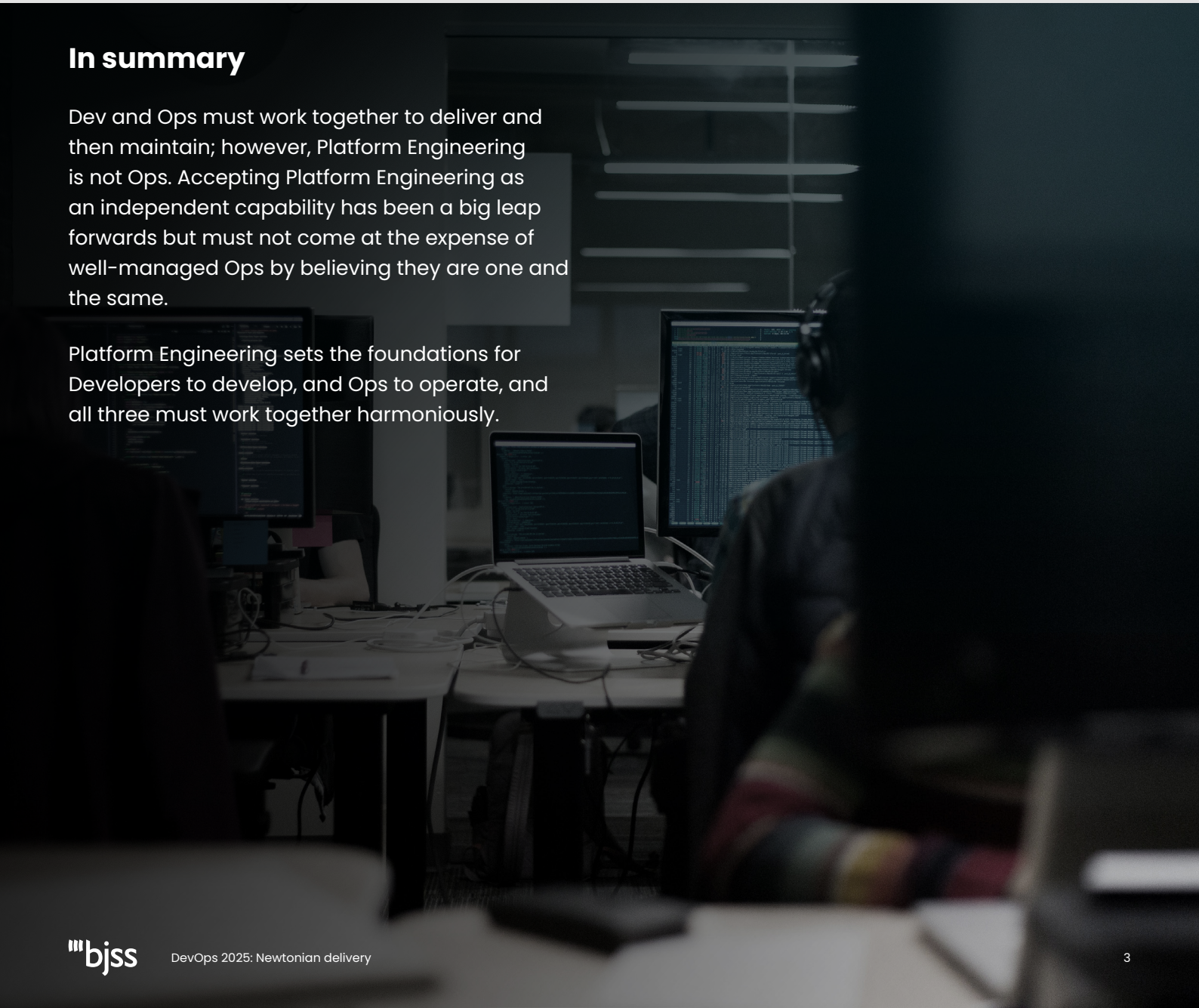
The development and platform backlogs are for tasks that drive change. They take the project from one state to another. Whomever is working on it, the operations backlog is for tasks that maintain the project in its current state. They keep the project running. They keep the project delivering value to the business.

This is where we return to the original emergence of agile DevOps. DevOps is a culture, not a role or a technical practice. For a time, DevOps was used to provide a working title for Platform Engineering, using a label that had been accepted in the C-suite to get the necessary investment.

Now that Platform Engineering is more established, it is time to brand it properly as a development activity. It is time to recognise that Software Development, Platform Engineering and Operations must be united but separate; each funded, resourced and managed appropriately to the value they provide.

Why aren’t we doing this already?

We live in a complicated and evolving world where success can be achieved in infinite combinations and can be extremely hard to measure. There are too many patterns to follow, too much guidance borne of limited experience. Too much risk associated with doing something different than has been done before. Cultural change takes time.



In summary

Dev and Ops must work together to deliver and then maintain; however, Platform Engineering is not Ops. Accepting Platform Engineering as an independent capability has been a big leap forwards but must not come at the expense of well-managed Ops by believing they are one and the same.

Platform Engineering sets the foundations for Developers to develop, and Ops to operate, and all three must work together harmoniously.

Challenge 2: Newtonian Platform Architecture

While Platform Engineering has become more established, and projects have accepted the need for Platform Engineers as a distinct role, Platform Architects are still a rare breed.

The role of Platform Architect is a critical one, responsible for the design of the platform, ensuring it meets the needs of the business and the Development teams. The Platform Architect must understand the needs of the business, the capabilities of the Developers and Engineers, and the capabilities and constraints of a rapidly-evolving suite of technologies and services. They must be able to design a platform that meets these needs and be able to communicate this design to all involved. They must be able to work with the Developers and Engineers to ensure that the platform is built to the design, and that it meets and continues to meet the needs of the business.

This role is not new, often found under the umbrella term "Solutions Architect", but rarely is it distinct and platform specialised. Usually, Platform Architecture is led by a Technical Architect with design authority and a background in Software Development, but performed by a combination of Developers and Engineers, each with their own perspective, priorities and experience.

This can lead to a lack of coherence in the design and can result in a platform that does not meet the needs of the business or the Developers.

Almost without exception it leads to unnecessary complexity. That complexity then becomes the basis for a litany of wasteful iterations each aimed at improving the development experience, some successful, some not, but ultimately spending a lot of time and effort iterating through one form of compromising complexity after another.

These architectural ownership gaps have become more visible since the recent explosion of the data and artificial intelligence (AI) domains. These each have their own unique requirements, and specialist knowledge is required to create efficient enterprise-ready solutions. There are many untrodden paths that mean production services can still feel like proofs-of-concept. Many organisations have begun to invest in the domains but have not accounted for the specialised experience required in Architecture and Design or are not yet able to find it.

What is the solution?

Architecture is not a singular function. To be effective, a project "Technical Architect" must have extensive knowledge in many specialist domains, making it incredibly difficult to grasp sufficient depth in each. Relying on the input of Engineers in each field can result in a patchwork of compromises, rather than a cohesive and unified design.

An enterprise project may require a team of Architects, each with their own specialism and responsibilities. For example, in a large-scale project with 100 Engineers spread across multiple domains, an Architecture team may comprise:

- **Enterprise Architect** - Overall vision and strategy of the project
- **Application Architect** - Design of the business logic being implemented by Software Developers
- **Platform Architect** - Design of the platform, including cloud and on-premises infrastructure as well as software development lifecycle tooling such as CI/CD pipelines
- **Data Architect** - Design of the data storage and processing systems
- **AI Architect** - Design of the AI models and the systems that run them
- **Security Architect** - Design of the security systems that protect the project, ensuring clarity and standardisation of compliance across all domains

These domains are usually already recognised as requiring oversight but fail to allocate direct technical responsibility and accountability for detailed design to the individuals. Each Architect role may be filled, but by an individual skilled primarily in management and policy, rather than the technical details and nuances of their domain as it applies across all the relevant technologies; often generating complexity and reducing quality, efficiency and flexibility.

In some domains, security being a frequent example, there will be an oversight team. The team will be responsible for the security of the project but will work at the microscale with Engineers in each domain to ensure that the security requirements are met, rather than at the macroscale to implement a coherent security strategy across the project aligned to the technology and the way it is being used.

Giving the responsibility of Architecture to an Engineering team, rather than to individual leaders in each domain can lead to a lack of coherence in the design and can result in inefficient and ineffectual solutions that are less cost effective or efficient than not having the solution, while also difficult to maintain and extend.

Specifically concerning the platform domain - one that is frequently not acknowledged as an independent architectural concern - the challenge is to recognise the need for a distinct Platform Architecture function, and to invest in the skills and knowledge required to fulfil the role. This does not require an overhaul of existing teams or significant resource investment, but a change in mindset, recognising the importance Architecture plays in a successful delivery, and reorganising to incorporate a distinct Platform Architect role.

In a team where the most experienced Engineer is acting as a de facto Platform Architect, the role of Platform Architect can be created by giving them the authority and responsibility to design a coherent and efficient platform, and to work as a peer with stakeholders and other Architects to ensure that the platform meets the needs of the business and the Developers, while remaining as simple as possible.

In a team where there is no clear leader in the platform architecture domain, it should be acknowledged that a gap exists. It's not necessarily a case of investing in a full-time Platform Architect role. Short-term consultancy may be sufficient to provide the necessary guidance and direction, and to help the team to develop the skills and knowledge required to achieve the desired outcome.

In a team where the architecture is being led by a generalist Technical Architect, it should be acknowledged that the Technical Architect may not have the necessary skills and knowledge to design a coherent and efficient platform. The Technical Architect may be able to provide valuable input but may not be able to design the platform on their own. In this case, the Technical Architect should be supported by a Platform Architect, who can provide the necessary skills and knowledge to design the platform, and to work with the Technical Architect to ensure that the platform meets the needs of the business and the Developers.

Why aren't we doing this already?

Agile has been a necessary revolution in Software Development, but it has also led to a lack of focus on Architecture, and short-sighted decision making. Agile has led to a focus on delivering features quickly, rather than on delivering a coherent and efficient solution.

Any upfront investment in Design can be seen as bad practice because it is "Waterfall" and therefore not "Agile". In truth neither is a tautology and even "WAgile" can be misdirecting if it is seen as a conjunction of two separate practices rather than a single practice that selects the appropriate elements of each.

Even within the pure-Agile mindset, Platform and Software Development have unique needs. There are many software development projects for which pure scrum is entirely appropriate, but for the platform aspects of the project, it is not. Platform timescales are massively impacted by external factors, and often many tasks are akin to mini projects. Trying to break work into time-bound sprints and measure value via sprint-velocity can, and often does, cause friction and box-ticking rather than accurate progress measurement. This has been a key factor in the problems experienced by teams and Engineers labelled as "DevOps" teams, where the platform is seen as a subset of the Software Development.

Where software changes can be made arbitrarily, akin to rearranging furniture or even erecting or knocking down walls, platform changes are usually more like changing the foundations of a building or preparing new foundations for an extension, frequently while the building is in use. They are not arbitrary, they are not quick, and they are not easy to undo. They require careful planning, consideration, and design.

Possibly the most impactful issue facing agile projects over the last decade has been a lack of consistent direction. Solving problems in isolation, without adherence to a common long-term goal has led to arbitrary tool selection, CV-driven development, and a lack of congruity in design. The result has been unnecessarily complex and inefficient solutions, held together by bespoke, fragile automation. The absence of this direction comes from abandoning "Big Design Up Front" and "Waterfall", and instead "empowering the teams" but the result is often a lack of any design at all.

The problem with Waterfall is not the design, it is the lack of iteration. The problem with Big Design Up Front (BDUP) is not the design, it is the lack of flexibility. There is nothing wrong with creating a design, the problem is in creating a design that is inflexible and unchangeable.

The solution is to create a design that is flexible and adaptable, that can be changed as the project progresses, and that can be iterated upon to meet the changing needs of the business and the Developers. So long as all involved are working towards a common goal that is well understood, changing the goal is relatively easy, having the same impact on everyone, all making the same changes in the same direction. This principle applies as much to simple matters like coding standards as it does to project architecture. XKCD humorously illustrates a common complaint about standards: **"There are 14 competing standards, we need one universal standard, now there are 15 competing standards."**

The joke highlights that each of the 14 existing standards aimed to solve everything, and introducing a new one only exacerbates the problem. The real solution, though contrary to human nature, is to adopt a single standard that everyone agrees on, whether it's an existing one or a new one.

For instance, if decided that "all strings should be single quoted," this is entirely reasonable so long as all existing code is updated. However, if the project's priorities, funding, or timelines do not permit the change to existing code, then it should not be applied for new code either. If all existing code adheres to a different standard, the change might be straightforward and automated. But with a mix of standards, it could become complex and time-consuming; especially if you have automation that relies on assumptions about the standard you follow.

Consistency is the key. The solution isn't to have no design, or to allow competing designs, but to have a single design that everyone follows. If change is justified, then make the change. If the change is not justified, then don't make the change. Don't make the change in one place and not in another just to address today's problem. While a team can attempt to manage consistency via peer review, only one person ought to be accountable for ensuring it is maintained.

In summary

For each distinct specialist technical capability:

- Have a “target design” that is well understood and agreed upon, that is aligned to the other capabilities
- Make one person responsible for that vision who has the knowledge and experience to design a coherent and efficient solution within the problem domain
- Give them the authority to make decisions and the responsibility and accountability to ensure that the vision is realised
- Delegate the role to another person in the case of absence rather than distributing the responsibility and risking inconsistency
- Make sure everyone knows the vision and is working towards it
- Treat change as expected and normal, so long as it is justified

If this is done correctly, then there is a significant hidden benefit. Domain-specific architecture design for a project does not have to be a full-time role. At the beginning of a project, it is a critical function, but as the project progresses, the need for the role diminishes. Ways of working become attuned, architectural norms become established and change frequency reduces.

The role can be reduced to a part-time role, or even a consultancy role, and the project can continue to function effectively. The role can be reactivated when a significant change is required, or when the project is at risk of losing its way. In the meantime, the person who filled the role can either move on to another project, or be assigned to multiple projects, providing the same service to multiple teams.

This can be a further benefit, allowing the person to bring their experience from one project to another, and to help multiple projects avoid the same pitfalls and to achieve the same successes, with standardisation and consistency across the organisation.

The role of the domain-specific architect adheres to Newton’s first law of motion. The role is the unbalanced force that drives change. When stable and consistent, the project may continue in the same direction at the same velocity, requiring intervention only for course correction.

Challenge 3: Newtonian Platform Engineering

Focus on what you do best and outsource the rest

Now that the modern project has recognised the need for a distinct Platform Engineering function, the next challenge is to determine how to deliver this function effectively. The Platform Engineering function is a complex and multi-faceted role, requiring a wide range of skills and knowledge.

In many projects however, the Platform Engineering function has very little relationship to the core business of the project. The Platform Engineering function is a support function, providing the infrastructure and tooling that Developers need to deliver the project. The Platform Engineering function is not the core business of the project nor the reason that the project exists, it is a means to an end.

When Platform Engineering is not the core business of the project, as little of the Platform Architecture as possible should be bespoke. As far as is reasonable and possible the platform should be:

- Standardised and generic, to reduce the complexity and the cost of the platform
- Simple, to reduce the risk of failure and the cost of maintenance
- Efficient, to reduce running costs
- Secure, to protect the project from malicious intent and accidental harm

Achieving this is difficult in isolation. When no standards exist, they must be created. When technology has not been selected, it must be evaluated, selected, configured and implemented. When no software development lifecycle (SDLC) processes exist, they must be decided upon, and tooling must be implemented to support them, and so on for infrastructure as code (IaC), continuous integration (CI), continuous deployment (CD), monitoring, logging, security, compliance, etc.

As a consultancy with over 30 years in the industry, BJSS has a wealth of knowledge and experience of the technologies and patterns available to solve common problems, and to deliver projects. Without this, one might be reliant solely on publicly available examples and vendor-provided solutions, which even in the best cases often do not integrate or function well outside of very tightly bound ways of working.

While different projects have different core technology requirements and commonalities that benefit from shared learning, what is more important are patterns that can be applied wholesale for frequently encountered situations. These patterns can be applied to the platform, the software development lifecycle, the data, the AI, the security, and so on, putting together proven approaches known to be suitable for enterprise project deliverables.

What is the solution?

Platform as a Product

Let us take for example a project that has a simple requirement: "Deliver a serverless web application that exhibits the company's brand, provides details about the company, and allows users to log in and view their account details".

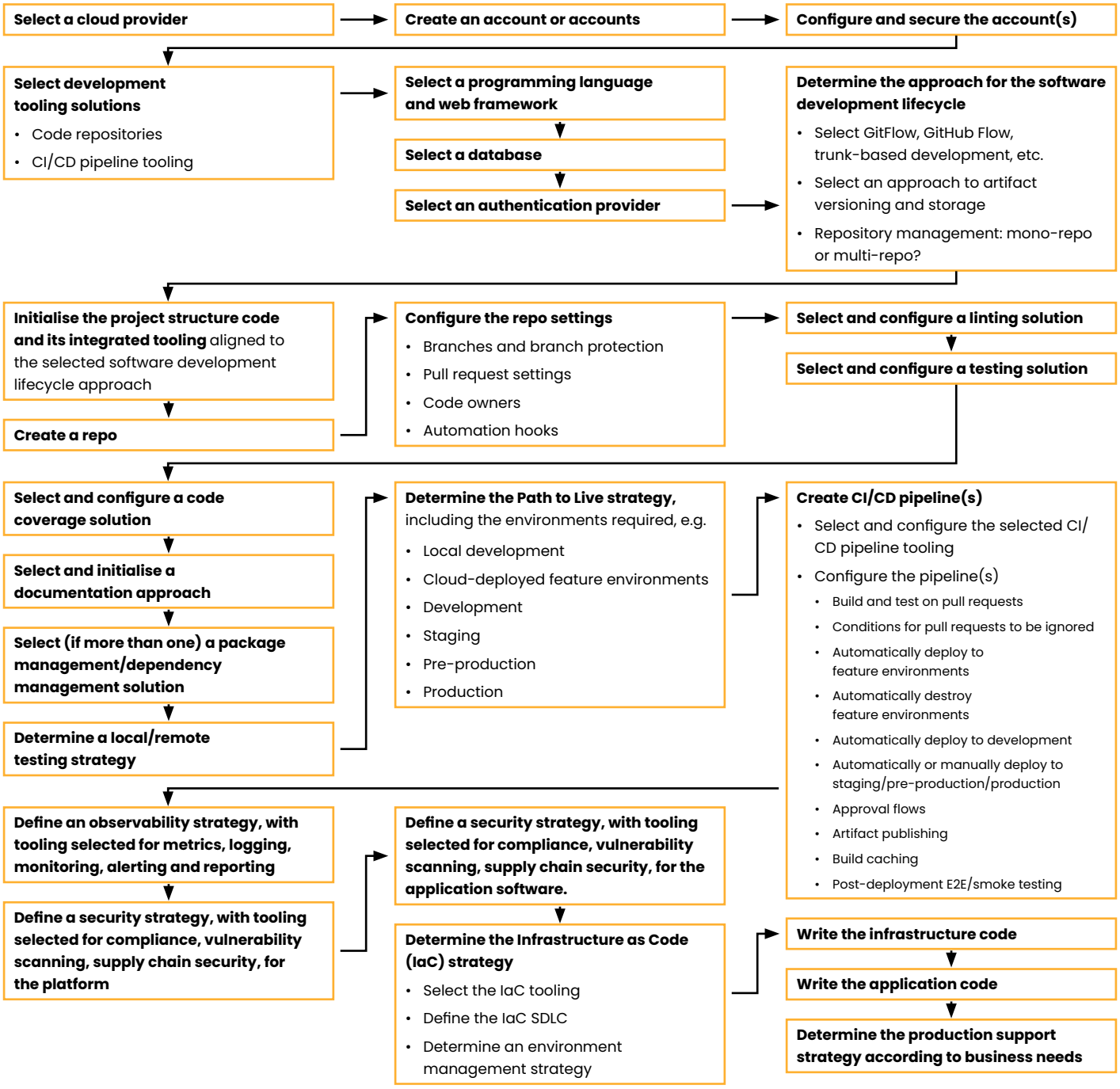
What are the initial steps normally taken to deliver this project?

This list does not cover all the steps required to deliver the project, but a significant portion of the basics. The question is, how much of the work is necessary to deliver the project? How much of the work is unique to the project?

It could be argued that of the below steps, only the following are wholly unique to the project:

- Write the infrastructure code
- Write the application code
- Determine the production support strategy according to business needs

Given mostly reusable infrastructure code for such a common situation, and a generic CMS-based application, even these tasks could be stripped bare.



Depending upon the project needs, the following may also be unique:

- Select a cloud provider
- Select a programming language and web framework
- Select a database
- Select an authentication provider
- Determine the Path to Live strategy, including the environments required.

The business/project may choose to make even more bespoke decisions but must accept that every one of them is a cost, a potential source of delay and likely not relevant to the core business of the project. Every other step is something that could be delivered in complete and ready-to-use form, as an accelerator. Any part of that accelerator could be tweaked to make the most of the possibilities of the technology, but the core of the accelerator would be the same.

For every aspect of the accelerator that requires no bespoke work, no platform engineering is required. Once it is delivered, the project can focus on the unique aspects of the project. The only Platform Engineering resource required is to deliver remaining bespoke work, to further expand or customise the accelerated platform, and to support application-specific infrastructure needs.

This then can be seen again as an application of Newton's first law of motion. The Platform Engineering resource is the unbalanced force that drives change. Once the accelerator is delivered, and tweaked as desired, the project may continue in the same direction at the same velocity, requiring intervention only for course correction - except of course application-specific bespoke infrastructure needs that cannot themselves be accelerated or templated.

The additional conclusion we can draw here is that if the Platform Architecture function is properly fulfilled, and the generic aspects of the platform are delivered with an accelerator, then much of the Platform Engineering function can be delivered as a product. An initial deliverable is needed to correctly implement the accelerator, but maintenance and support can mostly be handled as an in-sprint developer or operations task, with the Platform Engineering service acting mostly as a support function.

The more generic and simple the project, the less ongoing Platform resource requirement there is, allowing limited resource to be focused on the most complex and bespoke problems.

Why aren't we doing this already?

Whilst organisations are starting to establish Platform Engineering teams, they are not maturing processes and ways of working to extract value from the investment. The team can - and should - be concurrently building platforms as well as creating accelerators to address the wheels that are most commonly and unnecessarily being re-invented in their organisation.

As a consultancy, BJSS has over a decade experience of Platform Engineering and is perfectly positioned to help organisations these patterns and practices, and to share accelerators that will deliver the most value to organisations.

Combining knowledge of existing projects and leading-edge industry knowledge, BJSS has identified the most common and most costly problems and creates the accelerators that will solve them.

In summary

The industry has become lost in the diversity of solutions that achieve the same goal, and in putting in time, effort and resource to do the same thing - slightly differently - everywhere. Our experience of Platform Engineering at BJSS has led us to think bigger when it comes to undifferentiated heavy lifting.

We do more than just establish a secured location for development to happen. We commoditise our experience as we identify the patterns that are being unnecessarily repeated. For some projects we can make the entire software delivery lifecycle a plug-and-play experience. By doing so, we can implement a more hands-off approach to Platform Engineering; focusing resource on solving problems that haven't been solved before, instead of the ones that have.

You can find out more about BJSS on our website, or get in touch here.