Modern data engineering playbook

thoughtworks

Introduction

Data has been dubbed ‘the new oil’ – highly valuable, and able to power entire industries. But unless it’s handled carefully and strategically, it can be worthless or even detrimental to your business.

No matter what business you’re in, data can give you a clear point of difference and competitive advantage. And now that it is also the by-product of every digital step we take, your teams have the ability to collect and store more data than ever. Using it to make better strategic decisions, and create tailored and frictionless customer experiences.

However, as our need for insights at speed increases, centralized data platform architectures are failing to keep up. They lack the flexibility to enable dispersed teams to make informed and timely decisions. With the added challenges around governance, privacy and security and data quality, organizations are struggling to manage the complexity of operationalizing their data assets.

The modern data stack (MDS) – a collection of tools and patterns used for data integration – has emerged to address these challenges. It helps you analyze data, improve efficiencies, and unearth new opportunities. And there is a growing array of off-the-shelf tools to choose from, avoiding the need for custom-built solutions. In turn, that enables lower costs and increased time to production.

But there is more to reaping the benefits of MDS than having the right tools. Its success is underpinned by modern engineering practices and software delivery principles that allow you to accelerate development, de-risk large projects and continuously improve products. You also need the right skillsets across your teams, so they can make data more accessible, target the right problems and build the right products. And of course, valuable data intelligence relies entirely on the quality of data.

These are the things we address in this playbook, helping you save time, reduce risks, and improve the return on investment of your data projects.

Learn how shifting to a data product mindset will help you build the right thing and build the thing right – and how to assemble the right team to make it happen. Explore practices and principles that will speed up production, and find out how to save time by catching data quality issues early. And discover how you can embed security and privacy from the start to improve the quality of your product, build trust with your customers and allow you to move faster.

Let’s get started.

Shifting mindsets: why you should treat data as a product

By Keith Schulze and Kunal Tiwary

Today, organizations are increasingly recognizing the potential value of data – yet many fail to realize a return on investment from their data assets.

When it comes to developing data assets, many organizations take a 'build it and they will come' approach. While this philosophy may have paid off for Kevin Costner in Field of Dreams, organizations might not be so lucky. Because there is one inherent shortcoming of the approach: it doesn't consider the needs of data users.

What if we flipped the mindset, and consider some valuable user-centric lessons from our product teams? What if we managed data as a product – not just an asset? We're seeing this shift in perception gain traction, allowing organizations to unlock more value from data projects.

Rethinking data

The data as a product mindset is one of the four principles of data mesh, a style of data management which decentralizes project architecture models. Data as a product treats the data users as customers, developing data products to bring them value and help them achieve their end goals. For example, if your customer's end goal is to reduce churn rate by 10%, you will need to start with that goal and work backwards – and develop a churn forecasting data product that will meet this need. Thinking of data as a product means putting those user needs at the heart of their design. It's designed to be shared – not controlled.

Data as a product treats the data users as customers, developing data products to bring them value and help them achieve their end goals.

Zhamak Dehghani, author of *Data Mesh, Delivering Data Value at Scale*, and founder of the data mesh concept, describes it this way; "Data as a product is very different from data as an asset. What do you do with an asset? You collect and hoard it. With a product it's the other way around. You share it and make the experience of that data more delightful."

Product thinking requires a deep knowledge and understanding of your customer. Your teams can then build for real world problems – and continuously develop products that offer more value.

Build it *right* and they will come

Many data products fail because they are a solution in search of a problem – for example, ingesting a new dataset into the data platform because 'someone' will find it useful. Adding more data does not necessarily solve a customer's problems – or provide them with value.

That's why it's so critical to start by knowing who your customer is and what is most valuable to them. What problems are they trying to solve? What's at stake for them if they can't use or access the data easily? Those customers might be internal or external – the key is to think beyond simply offering data sources, and expecting users to adapt or compromise the way they work to use it.

Unfortunately, there's no silver bullet here. It takes time to understand your customers and their goals, and involves real-world testing and constant refinement. And once you've solved that for one group of customers, how do you scale and expand this? Can you make those products reusable, satisfying the needs of a broader range of customers?

At Thoughtworks, we have adapted the Double-Diamond design process model to make sure that we build the right thing and build it right. This starts with identifying what a customer needs. We use a structured discovery and inception process to uncover these requirements for any new data product. We then apply a set of well-understood practices and tools that are known to deliver high-quality software and data.

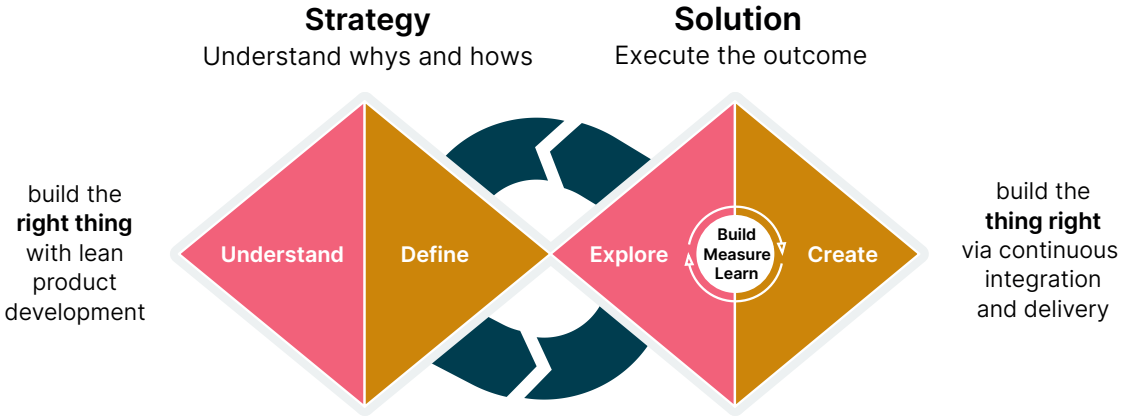


Figure 1: The Thoughtworks Double-Diamond design process model

"Data as a product is very different from data as an asset. What do you do with an asset? You collect and hoard it. With a product it's the other way around. You share it and make the experience of that data more delightful."

Zhamak Dehghani
Author of Data Mesh, Delivering Data Value at Scale, and founder of the data mesh concept

Encouraging ownership and responsibility

If, in the more traditional mindset, projects end once a dataset or report is delivered, product thinking requires teams to retain ownership over a data product for its entire lifecycle. That means data product owners are responsible for evolving and adapting the data product to ensure it continues to meet the needs of the customer even as their requirements change.

Much like software products, data products also benefit from a responsible and accountable team who continuously improve performance and release new features in a safe environment. It also reduces the feedback loops needed to evolve or enhance these products. It encourages direct communication between the producer and the consumer of data products – cutting out lengthy and convoluted central planning processes.

The owners of a data product are also accountable for maintaining agreed levels of service. This is important because without clear accountability, there might be complex processes and competing priorities to contend with when services go down.

Data product owners are responsible for evolving and adapting the data product to ensure it continues to meet the needs of the customer even as their requirements change.

For example, retail organizations use a number of metrics to facilitate demand planning (e.g. forecast accuracy, order fill rate). Different teams depend on these metrics to forecast and provision stock to meet the demand. Any delays or errors in reporting can have severe impacts to downstream business processes, leading to unhappy customers and a loss of revenue or a surplus of stock with a cost to business.

As a business evolves, there may be other demand planning metrics that would allow for more accurate forecasts; any delay in implementing these also means a sacrifice in potential profit. Businesses need to continuously evolve their demand planning process to use the most accurate metrics – and ensure that the metrics are reliable and high quality. Any error should be fixed promptly to minimize the impact on downstream consumers.

Having a team with clear accountability to develop, evolve and maintain these metrics as a product with high levels of service ensures that:

1. You're always providing downstream consumers with the most accurate metrics to facilitate demand planning, and
2. You minimize the impacts of outages on other teams in the demand planning process.

What it takes to make the change

A mindset shift such as this often requires cultural and behavioral change as well. If your organization wants to reap the benefits of user-centric data products, you will need to move to a more product-centric, customer-focused culture – and build cross-functional teams to support this approach.

Moving away from teams aligned to archetypes or skill sets, to small product-oriented teams with tightly focused goals is one way to get there. These teams may require a blend of different capabilities – such as data engineers, data scientists, QAs and designers – to develop a product that meets the needs of customers.

Creating a culture where learning from failure is embraced and celebrated is also critical to the success of developing effective data products. Finding what doesn't work, or where friction points lie, allows teams to adjust their thinking and approach for future projects – and continually improve products and customer experience along the way.

Creating a culture where learning from failure is embraced and celebrated is also critical to the success of developing effective data products.

Putting data as a product into practice

An effective data product should be:

Discoverable: If you want a customer to use a data product, they need to be able to find it. Discoverability can take many forms, from a primitive list of datasets on an internal wiki system to a full-fledged data catalog. Irrespective of the implementation, catalogs should house important meta information about the data products such as their owners, source of origin, lineage, and sample datasets.

Addressable: Data products should also have a single unique address where they can be found. This makes it easy for your customer to find them – and reduces the time your data teams spend on helping people locate them.

Self-describing and interoperable: Data products should provide users and automated systems with metadata in a way that allows users to self-service. For example, a data product should expose metadata describing the data sources used to build the data product, and the schema and information about outputs of the data product.

Trustworthy and secure: For a customer to use a data product confidently, the product should commit to an agreed level of trustworthiness (or quality). This means defining a set of Service Level Objectives (SLO) and measurable Service Level Indicators (SLI) upfront – and implementing automated mechanisms to test and report SLI metrics regularly.

Secure and governed by a global access control: With the rapid rise in data breaches, it has never been more important to secure your data products and build in security. While registered data sets should be automatically discoverable to all customers, they should not be accessible by default. Users will need to request access to each dataset they need, with data owners granting or denying access individually, using federated access controls.

Reaping organization-wide benefits

Adopting a data-as-a-product mindset is an organization-wide exercise – it demands a shift in not only perspectives but also in culture and practices. It is certainly worth the effort. The principles of product thinking allow you to develop multiple data products that can be used within the organization, and ultimately help you form an effective and streamlined network of data products. And when it becomes embedded in your enterprise, it helps raise the bar for tech teams – supporting them to always think about creating value and working towards outcomes for every user.

Engineering practices to accelerate your data product delivery

By David Tan and Mitchell Lisle

Approaching the development of data products as you would approach building software is a good starting point. But data products are typically more complicated than software applications because they are software *and* data intensive. Not only do teams have to navigate the many different components and tools that are part and parcel of software development, they also need to grapple with the complexity of data. Given this additional dimension, it can be all too easy for teams to get mired in cumbersome development processes and production deployments, leading to anxiety and release delays.

At Thoughtworks, we find that intentionally applying "sensible default" engineering practices allows us to deliver data products sustainably and at speed. In this article, we'll dive into how this can be done.

Applying sensible defaults in data engineering

Many sensible default practices have their roots in **continuous delivery (CD)** – a set of software development practices that enables teams to release changes to production safely, quickly and sustainably. This set of practices reduces the risk of error in releases, reduces time to market and costs, and ultimately improves product quality. **Continuous delivery practices** (such as automated build and deployment pipelines, infrastructure as code, CI/CD and trunk-based development) also **positively correlate with an organization's software delivery and business performance**.

From development and deployment to operation, sensible default practices help us build the thing right. These practices include:

- Trunk-based development
- Test-driven development
- Pair programming
- Build security in
- Fast automated build
- Automated deployment pipeline
- Quality and debt effectively managed
- Build for production

As we will elaborate later in this chapter, these practices are essential in managing the complexity of modern data stacks and accelerating value delivery because they provide teams with the following characteristics which help teams deliver quality at speed:

- **Fast feedback:** Find out whether a change has been successful in moments, not days. Whether it's knowing unit tests have passed, you haven't broken production, or a customer is happy with what you've built.
- **Simplicity:** Build for what you need now, not what you think might be coming. This lets you limit complexity, while enabling you to make choices that allow your software to rapidly change and meet upcoming requirements.
- **Repeatability:** Have the confidence and predictability that comes from removing manual tasks that might introduce inconsistencies and spend time on what matters – not troubleshooting.

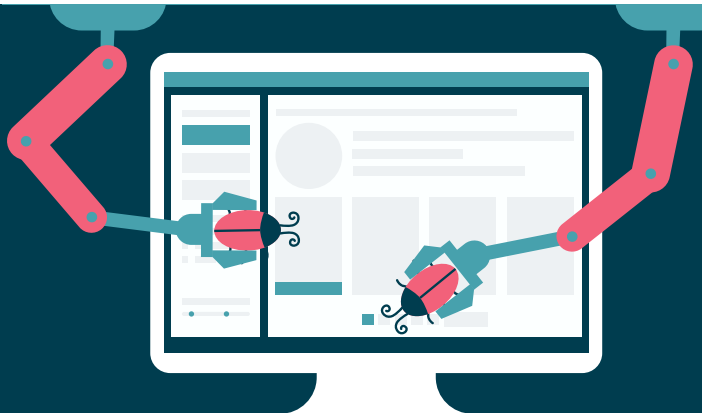
Engineering practices for modern data engineering

While there is a rich body of work detailing how you can apply continuous delivery when developing software solutions, much less is documented about how you can use these practices in modern data engineering. Here are three ways we've adapted these practices to build and deliver effective data products, fast.

1. Test automation and test data management

Test automation is the key to fast feedback, as it allows teams to evolve their solution without the bottlenecks that result from manual testing and production defects. In addition to the well-known practices of test-driven development (guiding software development by writing tests), it's also important to consider data tests.

Test automation is the key to fast feedback, as it allows teams to evolve their solution without the bottlenecks that result from manual testing and production defects.



Similar to the practical test pyramid for software delivery, the practical test data grid (Figure 1) helps guide how and where you invest your effort to get a clear, timely picture of either data quality or code quality, or both. The grid considers the following data-testing layers:

- **Point data tests** capture a single scenario which can be reasoned about logically, for instance a function that counts the number of words in a blog post. These tests should be cheap to implement and there should be many of them, to set expectations in a range of specific circumstances.

- **Sample data tests** provide valuable feedback about the data as a whole without processing large volumes. They allow you to understand fuzzier expectations and variation in data, especially over time. While they bring additional complexity and require some threshold tuning, they will uncover issues point tests don't capture. Consider using **synthetic** samples for these tests.
- **Global data tests** uncover unanticipated scenarios by testing against all available data. They're also the least targeted, most subject to outside changes, and most computationally expensive.

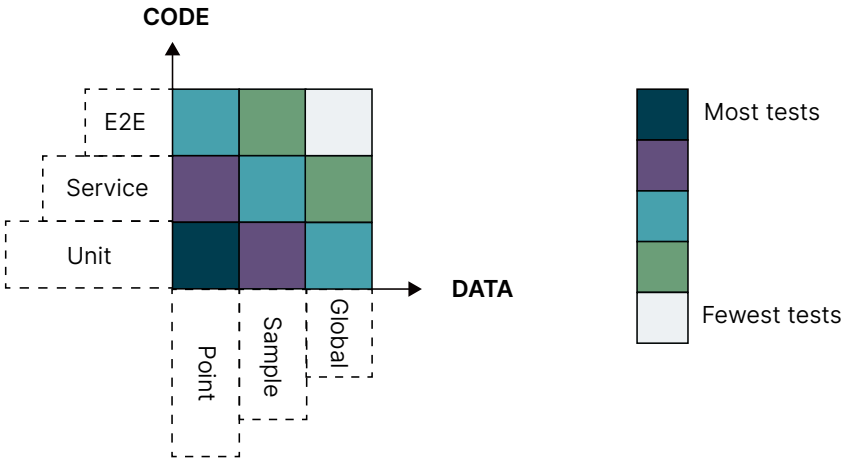


Figure 1: The practical data test grid

You can apply these tests to data alone or combine them with code tests to verify various stages of data transformation – in which case you would consider the two dimensions as a practical data test grid. Again, this is not a prescription, you needn't fill every cell and the boundaries aren't always precise, but this grid helps direct our testing and monitoring effort for fast and economical feedback on quality in data-intensive systems.

A word on test data management

You will need production-like data to use the practical data test grid. Start by thinking about **three planes of flow** (Figure 2):

- **In the code plane**, code flows along the Y-axis, from bottom to top, between environments (e.g. development, test, production). In traditional software engineering terms it's a CI/CD pipeline – the flow that software engineers are typically familiar with.
- **In the data plane**, data flows along the X-axis from left to right in each environment where data is transformed from one form to another. This is a data pipeline and is something data experts understand very well.
- **In the reverse data plane**, data flows along the Y-axis in the opposite direction of the code plane. You can create samples of production data into test environments using different privacy preserving or **obfuscation techniques** such as masking, differential privacy – or you can create purely synthetic data, using samples of production data.

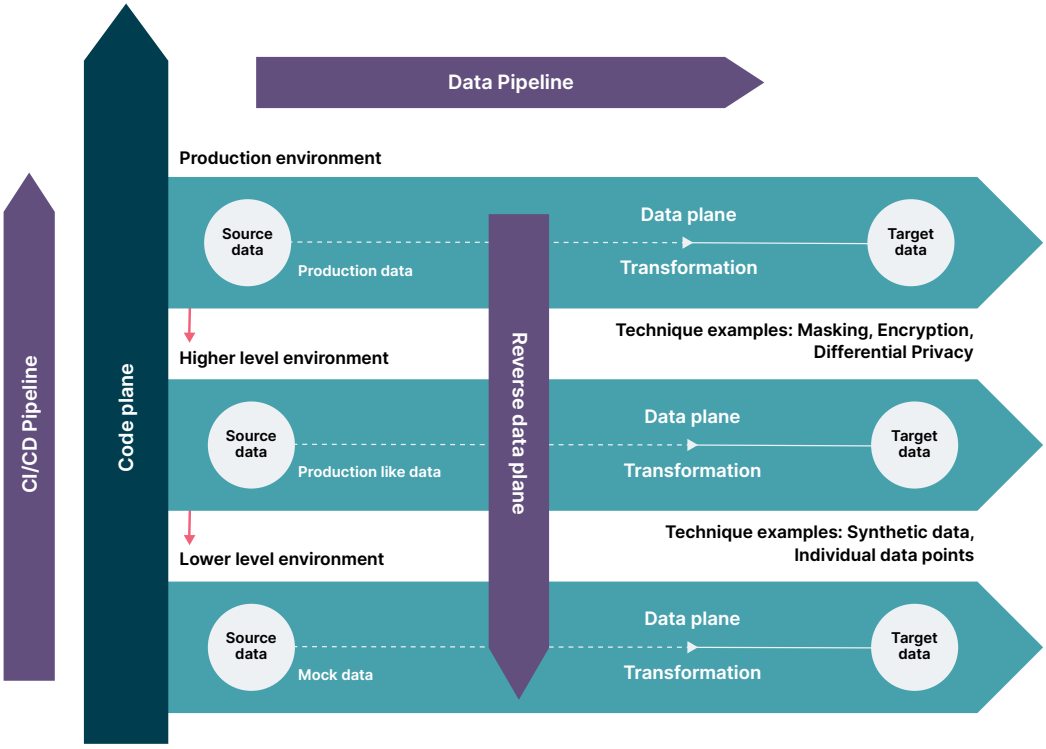


Figure 2: Visualization of a reverse data plane in a data pipeline

2. Case study: Accelerating delivery with sensible default engineering practices

Company Y is a large-scale software company whose customers use its software for reporting purposes. Company Y wanted to build a new feature that allowed its users to see historical as well as forecasted data that helped them make better informed decisions.

However, the historical data needed to train machine learning models to create forecasts was locked up in operational databases, and the existing data store could not scale to serve this new use case.

We applied sensible default engineering practices to build a scalable streaming data architecture for Company Y. The pipelines ingested data from the operational data store to an analytical data store to service the new product feature.

Sensible defaults in action

- We adopted **pair programming** where developers write code together and work on the new feature, providing each other real-time feedback through **test automation**. This involved unit, integration and end-to-end tests run locally.
- Pairs accessed **test data** samples that represent the data we expect in production and used git hooks that lint, check for secrets, run the test suites. If something was going to fail on continuous integration, the hooks shift feedback left and let the developers know before they pushed the code and caused a red build.
- We made sure code is always in a deployable state by adopting **continuous integration (CI)** – a practice that requires developers to integrate code into a shared repository several times a day. This sped up deployment, while a **fast automated build** pipeline running the automated test suites on the CI server provided fast feedback on quality.

- Our team built the artifacts once and deployed to each environment through **automated deployment**. We applied **infrastructure as code** (i.e. application infrastructure, deployment configuration, observability are specified in code) and provisioned everything automatically. If the build breaks in the pre-production environment, we either fixed it quickly (in 10 minutes or less) or rolled the change back.
- **Post-deployment tests** made sure everything was working as expected, giving us the confidence to let the CI/CD pipeline automatically deploy the changes to production (i.e. **continuous deployment**).
- **Observability** and **proactive notifications** allowed everyone on the team to know the health of our system at any time. And **logging with** correlation identifiers helped trace an event through distributed systems and observe the data pipelines.
- Where necessary, we **refactored** (rewriting small pieces of code in existing features) and **managed tech debt** as part of stories and iterations, not as an afterthought buried in the backlog.

Performing at an elite level

In just 12 weeks, we delivered a set of end-to-end streaming that continually updated the analytical store and powered the forecast feature that provided users with real-time insight. These practices helped the client team accelerate delivery and become an **"elite" performer**, as measured by the four key metrics:

- Deployment frequency: on-demand (multiple production deploys a day)
- Lead time for changes (time from code commit to code successfully running in production): 20 minutes
- Time to restore service: less than 1 hour
- Change failure rate: < 15%

For more complex changes, our team uses **feature toggles**, data toggles or **blue-green deployment**.

A note on trunk-based development

Throughout delivery, we apply **trunk-based development** (TBD), which is a version control practice where all team members merge their changes to a main or master branch. When everyone works on the same branch, TBD increases visibility and collaboration, reduces duplicate effort and gives us much faster feedback than the alternative practice of pull request reviews.

If we are averse to committing code to the main branch, it's usually a sign that we're missing some essential practices mentioned above. For instance, if you're afraid your changes might cause production issues, make your tests more comprehensive. And if you need another pair of eyes to review pull requests, consider **pair programming** to speed up feedback. It's important to note that TBD is a practice that's only possible if we have the safety net and quality gates provided by the preceding practices

In our next chapter, we'll look at how to build a team to help you build the right thing.

Data is a team sport: building an effective data team

By Keith Schulze and Kunal Tiwary

Consider your favorite sporting team: each person brings a unique set of skills and experience to the team and plays a specific role. Powerful and accurate, quarterbacks call the shots on the field, while linebackers defend with strength and speed. Versatile midfielders connect the two and keep the ball moving smoothly. To be a successful team, these roles and skills must work together seamlessly.

Data teams work similarly. As an integral part of larger processes, a successful team will bring together complementary skills, experience and knowledge to deal with critical questions such as:

- What new markets, services, or cost optimizations is the business embarking on?
- How could data help our people make these decisions?
- Is this data available to key decision makers in a form that is understandable?

As the modern data value chain evolves, organizations are making a fundamental shift in how they think about data – and how they build teams around it. As you focus on democratizing data across your organization, think about how you can align your internal structures around the people who make up your data teams. Here's what we have found useful in building data teams for specific products.

Forming effective teams

Splitting domains or areas of interest is a great way to start identifying what teams you'll need to form. Look for areas of high cohesion – where elements are closely related to each other and have a common purpose – and low coupling – modules that work independently of each other – between domains. For example, Netflix maps some of its domains and data products as follows:

- **Subscriptions:** forecasting churn and helping predict which customers are likely to cancel their subscription
- **Content:** content recommendations and ranking
- **Player:** client related statistics
- **Payment:** fraud detection

Much like software teams, data product teams own their products collectively. Each product should have a nominated product owner who acts as the team's ambassador and key communicator to stakeholders and other data product teams. They drive the product roadmap and lifecycle, communicating expectations and facilitating collaboration.

Elements of a winning data team

Developing effective data products requires a specialist team with a set of multidisciplinary skills, experience and knowledge, including data engineers, data scientists, data product managers, data UX researchers and analytics engineers.

One of the most effective ways to build your data team is to develop roles that focus on specific aspects of the product's infrastructure, development and lifecycle. Each role brings a different set of skills, strengths, and approaches needed to create value from data. The nature of the data product you want to build will dictate the roles you'll need.

Developing effective data products requires a specialist team with a set of multidisciplinary skills, experience and knowledge.

It's important to note that a role is distinct from an individual who plays a part in your team. In fact, some people may fall under multiple roles. You should consider the roles needed to build the product, and those that will operate and maintain it. Different teams will:

- Create data products and make sure they're delivered through reliable data pipelines
- Use data products and combine them with advanced analytics to create new business value
- Ensure data products are dependable and run smoothly

In some organizations it's also common to see platform teams with specialized knowledge in certain technical competencies such as infrastructure or data science. This helps reduce the cognitive load on a data product team, because team members don't need to be specialists outside of their skill sets.

Roles in practice

Consider a scenario where you need to build a new financial services offering for a retailer. Your goal is to serve aggregated customer credit history information to an external entity, safely and securely, to help them make credit lending decisions.

The **high-level requirements** include:

- Ingest credit history data from several internal data sources
- Transform and aggregate credit history data into a format that supports credit lending decision making
- Provide a secure API to serve real-time requests for the aggregated credit history information for a customer based on a unique customer identifier.

The **cross-functional requirements** include:

- Credit history data must not be older than one day
- Data should not leave a secure network environment (i.e. be on the public internet)
- Sovereignty and governance of customer credit history data must remain with the retailer. Any data transferred to the external lending provider should be audited and governed by policies for its reuse and handling.

Let's consider some of the roles you will need to build and evolve the customer credit history data product into the future.

- **Product owner:** The product owner is accountable for maximizing your data product's value. They also play an important role in supporting the team at key decision and prioritization points. They will help prioritize which insight should be built first based on the return of investment of the feature and the effort involved.
- **Business analyst:** they play a vital role in understanding and aligning the value of your data product with the needs of customers and the wider business.
- **Data engineer:** they build the pipelines that source data from several internal systems, and transform and aggregate the data into a form that supports credit decision making.
- **Infrastructure engineer:** they build reusable and scalable infrastructure around the data product to facilitate reproducibility, continuous integration/deployment and automate as much as possible.
- **Backend engineer:** they build business logic in the form of data APIs to make data integrations with UI, other products and visualisation tools easy.
- **Quality assurance:** accountable for maintaining data and product quality. This role is essential to build trust with the customer.
- **Data science:** depending on how well-defined the credit decision making process is and whether we know what data is required, this role may be required to help with this understanding.

An important note on security

Since you will be sharing potentially sensitive data across different organizations, security is a critical concern for this data product. While it's vital that there is someone in the team leading on security, it should nevertheless also be viewed as the whole team's responsibility – and you should build security into your product with support from a core security function in the business.

Remember, **you don't need dedicated people for every role**. You might have experienced team members who fit many roles, or some of your core platforms might play a supporting role. For example, you might ask someone to be a security and privacy champion and make them accountable for making sure that the team follows good security practices. They can work closely with a core security platform team to run important activities like setting security objectives or running threat modeling sessions.

The skills and knowledge required at each stage of the data product's lifecycle is different - so the roles you'll need will change along the way.

While it's vital that there is someone in the team leading on security, it should nevertheless also be viewed as the whole team's responsibility.

On the importance of soft skills

While there is overlap in skills in some data team roles, the importance of the below soft skills to round out the roles cannot be overstated.

Leadership: This is vital to establishing a culture of treating data as a product in an organization. Especially if an organization is transitioning from a centralized data team to a decentralized model where autonomous teams are formed around a data product. During this transition, there will be periods of uncertainty and questions about whether the return on investment justifies the effort. Having experienced leaders with a clear vision for how a data product will deliver value for customers and the business can help a team navigate through these uncertain times –and convey the long-term benefits of a data product to the wider business.

Courage to speak up: Speaking up can take many forms. For example, questioning why we are doing things that aren't aligned towards our values and goals. Or raising challenges and providing constructive suggestions in a team retrospective (assuming a psychologically safe environment) as the first step towards continuous improvement.

Communication and storytelling: Effective communication and storytelling help to make data work, which tend to be technical in nature, accessible to non-technical stakeholders, encourage collaboration and improve overall outcomes.

Aligning data teams to organizational structures

Decentralized data teams formed around data products should have their own C-Suite executive and be treated as part of mainstream engineering. Leaders need to set a direction and provide governance – enough to empower the teams and enable autonomy so they can freely experiment and discover. This allows them to deliver true value for customers. The executive's KPIs should also align with their data team's goals.

Defining what success looks like for your data team is critical – and goals should focus on enabling decisions (outcomes) rather than the number of dashboards built (activities). For the Subscription domains in the Netflix example above, a success metric might be to reduce churn by 10% over the next quarter – rather than measuring the number of data points acquired to produce that metric.

Holding data team sessions is a great way to communicate these goals and how they fit in with the rest of the organization and build internal understanding of the teams' functions and roles.

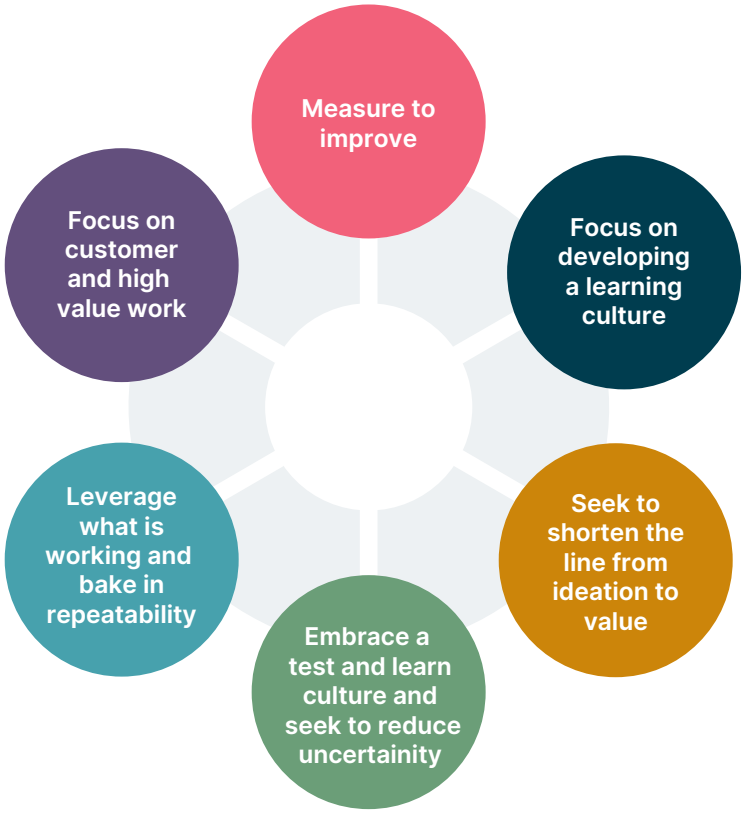


Figure 1: The culture and mindset of decentralized teams

As the managers and coaches of elite sporting teams know, an unbalanced team can fall short when it's time to perform. We've seen many organizations fail to realize the importance of balancing various skill sets when forming their data teams. This often leads to poorly engineered products or user experience – or turning products into a feature factory without considering the use case and customer value.

When you can draw on a broad range of expertise, and have the right structures to get the most value out of each skill set, your teams can target the right problems and thrive. As the famous saying goes, teamwork divides the task and multiplies the success.

Three delivery planning principles for iterating towards the right data product

By David Tan, Keith Schulze and Mitchell Lisle

In a previous chapter, we took a deep dive into the engineering practices that will help you deliver your data products quickly, safely and sustainably. Now we'll explore delivery planning principles to guide how we shape and sequence our work. These enable teams to add incremental value, de-risk large projects, and find opportunities for continuous improvement when delivering data-intensive products.

Note: We won't touch on typical iteration planning and release planning activities, such as story estimates and tracking cycle time, as they are becoming increasingly common in the industry. Those practices are still important, and we want to complement that by sharing additional principles and practices that provide an intentional focus on creating value.

Principle 1: Vertical thin slicing

One common pitfall in data engineering is the bottom-up sequential delivery of functional layers of a technical solution (think data lake, data warehouse, machine learning pipelines, and user-facing applications) – or horizontal slicing.

The downside of this approach is that users can only provide valuable feedback after significant investments of time and effort. It also leads to late integration issues when horizontal slices eventually come together, increasing the risk of release delays.

So, how can we plan and sequence our work to release value early – and often? How can we avoid the quicksand of data engineering for the sake of data engineering, and create a cadence of demonstrable business value in every iteration? The answer is by flipping our thinking, and slicing work vertically.

Thin vertical slices help to ensure that:

1. **At the level of stories**, you articulate and demonstrate business value in all stories, and ensure that the majority of stories are independently shippable units of value.
2. **At the level of iterations**, you regularly demonstrate value to users by delivering a collection of vertically sliced stories within a reasonable timeframe.
3. **At the level of releases**, you plan, sequence and prioritize a collection of stories that's oriented towards creating demonstrable business value.

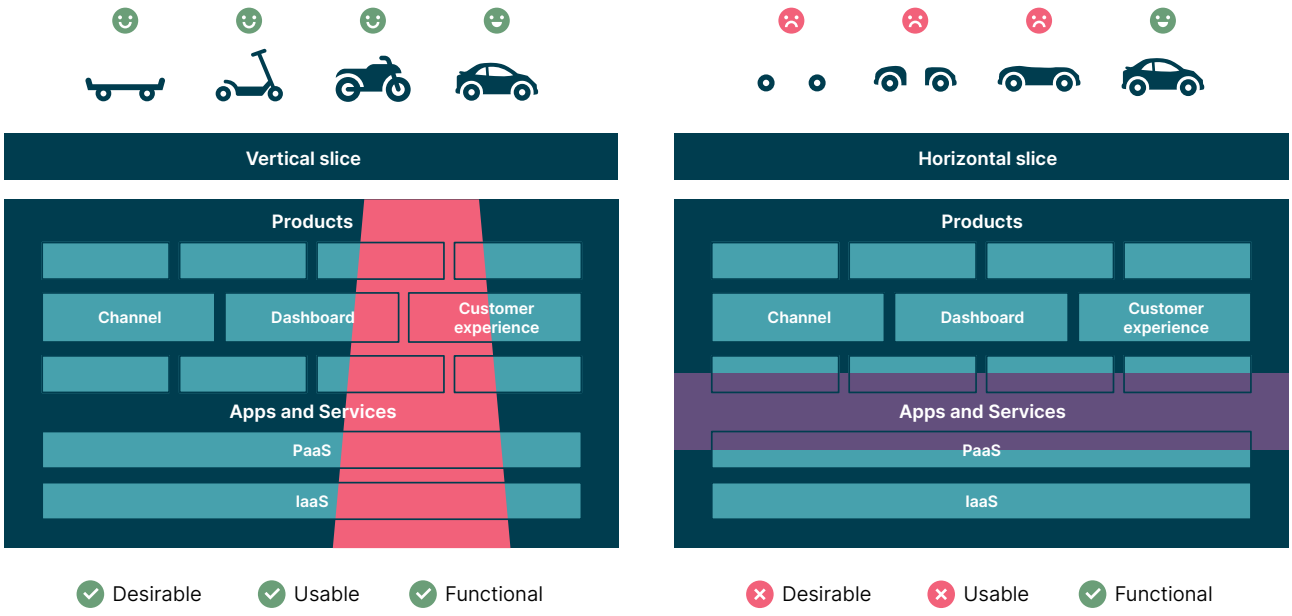


Figure 1: Delivering early with thin vertical slices

Tracing a thin slice of value through the components of a data ecosystem and delivering value in vertically sliced user stories enables you to test and learn more cost-effectively, solve customer problems more efficiently, and deliver value sooner. We will describe what this looks like in action in the case study below.

Principle 2: Data-driven hypothesis development

When embarking on data products, we often find ourselves solving problems with little data (that's why we're investing in data engineering!) and high levels of risks (the "known unknowns" and "unknown unknowns", see Figure 2). In this scenario, we should focus on finding the shortest path to the right solutions, and eliminating 'incorrect' solutions as soon as possible.

Figure 2: Four problem categories

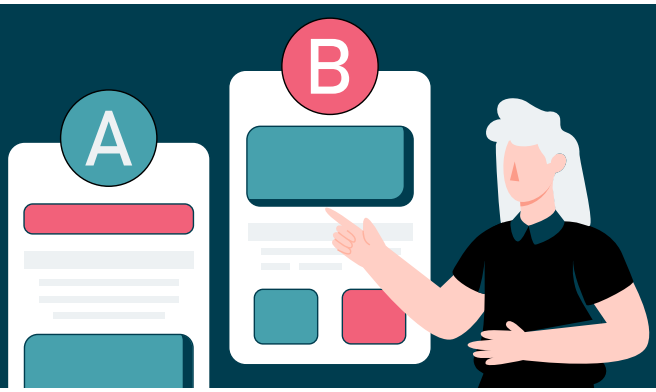
Data-driven hypothesis development (DDHD) is an effective way to approach these problems. Hypothesis testing is a powerful tool that can help to de-risk a large piece of work, and should be used not just before, but also during, delivery.

In essence, DDHD is about formulating hypotheses, running small experiments with clear outcomes and criteria and using the data we collect to tell stories and share your lessons with your team, business and stakeholders. The case study below will illustrate this in action, but for now this is what a hypothesis looks like:

- We believe that **<this capability>**
- Will result in **<this outcome>**
- We will know we have succeeded when **<we see a measurable signal>**.

DDHD creates a space and a framework for teams to run short experiments, to learn as we deliver value incrementally, and to continuously apply our lessons learned to have greater impact and reduce risks of costly and unvalidated investments.

Hypothesis testing is a powerful tool that can help to de-risk a large piece of work, and should be used not just before, but also during, delivery.



Principle 3: Measuring delivery metrics

Research has found that high-performing technology organizations perform well in four key metrics:

1. delivery lead time
2. deployment frequency
3. mean time to restore service
4. change fail percentage

The four key metrics provide insight into the flow and friction of value delivery, and they're a great starting point for identifying what is working well, and what needs to be improved. For organizations that do not have the platform tooling that allows teams to measure the four key metrics, you can get started simply by surveying teams regularly with DORA's quick check tool. Even though the precision is not as high, this method gives you a first indication of where your organization stands, and what the trends are.

In addition, you should also measure outcomes-oriented metrics that are relevant to your organization, such as improvement in efficiency and customer satisfaction. Outcomes-oriented metrics help to align teams towards activities that contribute to organizational goals rather than busywork. The diagram below gives an example of what outcomes-oriented metrics might look like in the insurance domain.

4 Key metrics of DevOps*		+3 Key metrics of excellence		4 Key business outcomes**	
Mean time to restore (Less is good)	On average, how long (hrs, days, wks or mnths) to restore services that made a product usable and useful.	**Build failure rate** (Less is good)	In the specified interval, what percentage of the builds failed due to any reason.	**Future sensing**	How many alternates insurance orgs. were identified. (More is good)
Change fail percentage (Less is good)	What percentage of changes to production had to be rolled back or hotfix/patched.	**Security warnings** (Less is good)	How many security warnings (of high, medium or low severity) are currently being reported by the underlying tool.	**Improvement in efficiency & effectiveness**	For a Delivery Analyst, mean time in identifying a processing claim issue. (Less is good)
Lead time (Less is good)	On an average, how long (hrs, days, wks or mnths) for code to reach production when committed.	**Tech debt** (Less is good)	How long (days, wks or mnths) it will take to rewrite the code that was the result of prioritizing speedy delivery over perfect code.	**Improvement in customer experience**	Number of issues reported by the processing agency. (Less is good)
Deployment frequency (More is good)	In the specified interval, number of times code was pushed to production.			**Increase in influence**	How often are new insurance orgs contacting the processing agency as their first option. (More is good)

*Based on the book Accelerate by Nicole Forsgren, Jez Humble, Gene Kim
**Example from insurance processing project

Figure 3: Measuring delivery metrics and outcomes-oriented metrics (for example: improvement in customer experience, improvement in efficiency) help teams to focus on impactful work, rather than busy work

However, these metrics should be undergirded by a healthy organizational culture and value-oriented mindset. Otherwise, we can fall into the trap of dysfunctional metrics. As Godhart's law states: "When a measure becomes a target, it ceases to be a good measure." And keep in mind Hawthorne effect – when your team knows they're being measured, there might be a reflex to bend the rules and find loopholes to meet the targets.

Case study: Applying delivery planning principles to shorten feedback cycles and create the right product

A B2B company, let's call it Company X, wanted to help its customers run and grow their business with a new financial service offering. Company X knew that by using the historical transaction data it had from its customers, it could offer a better lending experience than other financial service providers. So we worked with Company X to create a customer credit history data product that enables it to recommend suitable financial products to its customers. Customers that consented to Company X using their data for this product would have a smoother experience in getting an appropriate financing for their business retail purchases.

Vertical slicing in action

Company X did not have a data platform for extracting, processing and creating new data products – the data we needed to build this was siloed in transactional datastores in production.

Rather than wait for a data platform to be completed (i.e. horizontal slicing), we applied **lean product development and data mesh principles** to build a data product that provides real customer value in the short term, while supporting extensibility towards a shared platform in the medium to long term. This allowed us to deliver a low-risk solution fast (in just over 4 months), while also providing valuable insight into an ongoing data platform development effort.

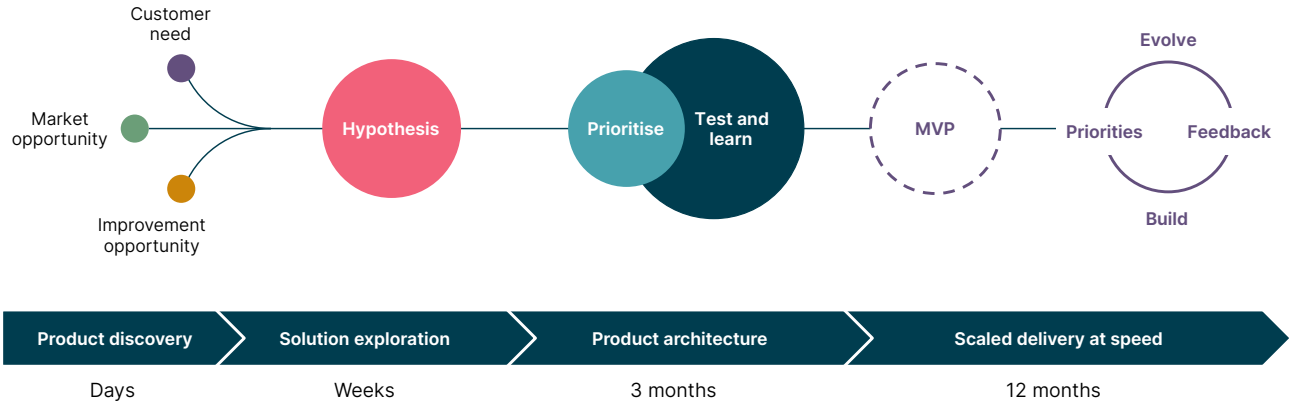


Figure 4: Build measure learn in context. A Lean approach to product delivery

Phase 1: Discovery

With the help of domain experts, we refined Company X's business requirements to outline data sources needed for the credit history product. We found we only needed seven (out of 150) tables from the source database to deliver the minimum requirements. This reduced data ingestion efforts, as we didn't need to process or clean unnecessary data. Over 6 weeks, we also refined the features and cross-functional requirements of the customer credit history data product, and aligned on the intended business value.

We articulated hypotheses to help us find the shortest path to the 'correct' solutions. These hypotheses helped us stay on the right track towards our goal of building the right product. For example, we could validate our approach by running an experiment and collecting data on one of our hypothesis:

- **We believe that** establishing an automated rule-based pre-screen based on various dimensions of a customer's transaction history
- **Will result in** a scalable way of identifying creditworthy customers
- **We will know we have succeeded when** an automated pre-screen application is able to reject non-creditworthy customers with a X% margin of error relative to credit assessments done by professionally trained domain experts.

Phase 2: Delivery

Once everyone was aligned on the product form and value it should deliver, we started developing a minimum viable product (MVP). Scoping an MVP can be difficult. We aimed for the **thinnest 'vertical' slice** that provided feedback about the viability of the data product, and ensured that it was close enough to the final product from a customer perspective to continually test our hypotheses. The MVP also uncovered potential edge cases, hidden or missed product opportunities, and possible obstacles. This early feedback helped to identify risks and where we can focus our risk mitigation efforts when further developing the product.

It also helped to define the data sources and transformations that we could leverage when iterating on future releases of the product. Our focus for the production delivery phase was to implement well-governed transformations on supportable and extensible data infrastructure – and serving the results to data product consumers. We applied our sensible default engineering practices, such as test-driven development (TDD), infrastructure as code, CI/CD, observability in both code and data planes, among others.

Delivering an independent, comprehensive data product

Within 10 iterations (over 4 months), we delivered a consumable, fully automated and comprehensively governed data product with no dependency on a centralized data platform. The team also measured the **four key metrics** (e.g. delivery lead time, change fail rate) and other **delivery metrics** (e.g. velocity, burnup rate, etc) to provide insight into how we're progressing towards the goal. The metrics helped us recalibrate delivery parameters where necessary.

Amplify impact, reduce time to delivery

These practices have helped Thoughtworks deliver value for clients time and again, and are sensible defaults that we bring to every data engagement to accelerate delivery and bring extraordinary impact. Wherever you are on your delivery journey right now, you can chart a path towards delivery success by:

- **Building awareness:** Are there any gaps or opportunities in your current delivery planning practices?
- **Being open to what needs to change:** How would you apply the principles and practices outlined in this chapter to help you improve your delivery planning?
- **Executing the change:** Connect industry-tested recommended practices with practical experience in successfully delivering data products.

In the next chapter, we'll share how you can save hours by better managing the quality of your data.

Architecture for data systems: how to balance trade-offs for technology decisions

By Simon Aubury and Kunal Tiwary

Data is an integral part of every product delivery and customer engagement. When designing data systems you need to understand both data and technology, while appreciating the ultimate value a product will bring to customers. It doesn't matter how big or small your data product is, establishing sensible defaults helps balance the trade-offs of particular technology decisions.

Assessing best practices around data management is a good starting point that can guide your data design choices. As data engineers, you should select technology tools that work well together for a project and are suitable for the broader needs of data across the organization. At its core, every technology decision needs to be driven by the value it will provide to the business.

In this chapter, we cover the baseline principles to get you started and some considerations for balancing the trade-offs of technology.

Default principles

Given today's complex data and technological landscape, the notion that there can be a single best way of doing anything seems ambitious. But sensible default practices are a great starting point because they're an effective way of building architecture on shared values. They also allow you to be technology agnostic, while focusing on the elements of good design. And for data projects, they can provide an initial set of baseline principles: effective practices and techniques to get you started.

Figure 1: Data default practices (In addition to core engineering sensible defaults)

As with software projects, we believe it's essential to establish best practices around data management for modern data platforms, including:

- Data quality
- Capacity and performance planning and measurement
- Incremental value delivery
- Observability
- Security and compliance
- Discoverability
- Ethics and bias
- Tracked measured and reproducible experiments
- Measuring architectural fitness

However, there may be circumstances which make the default choice invalid — or at least suboptimal. For example, if your product differentiation requires ultra low latency reads at the expense of consistency, you will need to use a specialized niche data store for your use case.

Measuring architectural fitness

Data architecture needs to grow and evolve with the needs of the organization. Evolutionary architecture is an approach that enables architecture to change incrementally – allowing your business to respond to new demands quickly. To make sure change doesn't compromise quality or cause architectural issues, measure how the architecture meets the original use case over time. Fitness functions provide an objective measure, informing the development process as it happens, rather than after the fact.

Some fitness functions for data architecture to consider include:


Cost


Data latency


Data Volume

Putting use cases before technology

There's often a fixation on labelling a data problem as being a transactional or analytical workload, or a use-case as being a real-time or batch system. This often leads to characterizing a business problem as "suitable" for a technology. However, technology is there to support the business, not the other way around. Take data processing as an example.

Waiting until the end of the day to process data in a batch is a bit like buying a newspaper to find out what happened in the world yesterday.

Business processes are akin to a flow of events – and virtually all data you deal with is streaming in this flow. Data is almost always produced and updated continually at its source, and it is constantly arriving. Waiting until the end of the day to process data in a batch is a bit like buying a newspaper to

find out what happened in the world yesterday. While for some use cases, such as billing and payroll systems, this is acceptable, others require more immediate streaming data processing.

When designing appropriate architecture, focus on developing solutions that process data to meet the business outcome you're working towards. You need to step back and appreciate what the architecture is there to support. Are you building a system to facilitate "transactions" (think sales on an ecommerce website, or a payment processing system)? Or are you trying to "analyze" history to identify trends and use aggregated data to draw insights? Start with the problem you are trying to solve. Then look at the characteristics of the relevant business data, and consider how technology and project architecture might be able to better support those workloads.

While it's crucial to use the right data store for the right use case, you want to solve the problem, not build to the constraints of the technology. The wrong technology choices can misdirect engineering effort and undermine the likelihood of future business success.

Balancing the trade-offs of technology

Technology changes fast; this is especially true for data systems. The technology you select needs to meet the increasing demands and expectations for data platforms, address a wide range of needs – from transactional and operational to analytical – and enable interactive data exploration in real time.

But finding the right technology can be time consuming. To speed up the process, Amazon founder, **Jeff Bezos suggests** you don't deliberate over easily reversible, "two-way door" decisions. Simply walk through the door and see if you like it — if you don't, go back. You can make these decisions fast and even automate them.

Few, if any, data decisions are hard to reverse. But those that are need to be made carefully. A **data technology radar** is a great way to balance risk in your technology portfolio and pollinate innovation across teams, experimenting accordingly. It allows you to work out what kind of technology organization you want to be – and objectively assess the data tools that are working and those that are not.

It's also important to only invest in and use custom-built solutions where it is a differentiator to the business or provides a competitive advantage. Consider what's important to your business:

- Security, schemas, design and lineage – the non-negotiables for system design
- An upfront tiring of latency, correctness and durability – deciding the ranking is still important
- Batch, harmonize and consolidate delivery – because costs ebb and flow

Moving fast may lead to duplication, while consolidating effort on a centralized data platform will eliminate duplication but take longer. You need to understand the trade-offs and make data architecture decisions at the enterprise level quickly. Slow decisions can cause your data infrastructure to unnecessarily proliferate which can be difficult to reverse if you want to consolidate your infrastructure.

Optimizing for sensible data defaults can make it easier to select the right technology and make good architecture choices. It can also help you validate whether your investment is providing the business with the competitive advantage you expect it to – helping you make more informed decisions.

In the next chapter, we'll share how you can implement an effective data strategy that provides the foundations for managing **data quality**.

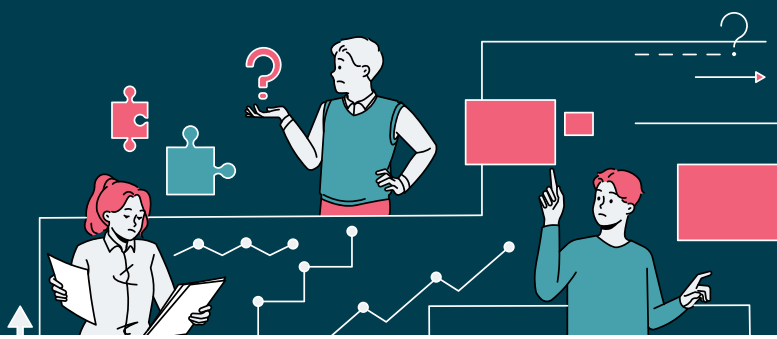
Quality is king: finding the value in your data test strategy

By Simon Aubury and Kunal Tiwary

Imagine what your data teams could achieve with an extra two days per week. The thought is exciting, isn't it? But where would this additional time come from? The answer may lie in better managing the quality of your data. The best way to do that is by catching issues early with the help of a rigorous data testing strategy.

The impact of moving data testing upstream shouldn't be underestimated. Research suggests that data teams spend **30-40% of their time focusing on data quality issues**. That's a significant amount of time they could be spending on revenue-generating activities like creating better products and features or improving access to faster and more accurate insights across their organization. And beyond productivity and organizational effectiveness, data downtime – caused by missing, inaccurate, or compromised data – can **cost companies millions of dollars** each year and erode organizational trust in your data team as a revenue driver for the organization.

Research suggests that data teams spend 30-40% of their time focusing on data quality issues.



Missing values in datasets can lead to failures in production systems, incorrect data can lead to the wrong business decisions being made and changes in data distribution can degrade the performance of machine learning models. Irrelevant product recommendations could impact customer experience and lead to a loss of revenue. In sectors such as healthcare, the consequences can be far more significant. Incorrect data can lead to the wrong medication being prescribed, triggering adverse reactions, or even death.

In this chapter we'll take a look at how to implement an effective data strategy. But first, we'll look at the core considerations that lay the foundations for data quality.

The considerations and trade-offs of data quality

Instilling organizational trust in the quality of its data and the data team is vital. However, this can only be done by describing what we actually mean by data quality: what features and dimensions are fundamental to it. Here are five areas to consider:

Freshness: The importance of data recency is context specific. A security monitoring or fraud detection application requires very fresh data to make sure aberrations are detected and can be dealt with quickly, whereas training a machine learning model can tolerate more latent data.

Accuracy: Life impacting decisions such as drug effectiveness and financial decisions have little room for inaccurate data. However, we may sacrifice accuracy for speed when offering suggestions on a retail online store or streaming service.

Consistency: Definitions of common terms need to be consistent. For example, what does "current" mean when we talk about customers – purchased last week or two months ago? Or what constitutes a "customer" – already signed on, authenticated, a real human?

Understanding the data source: The source of data or how data is captured can affect its accuracy. If a customer service representative at a bank selects a drop-down field in a hurry or without validation, a manual error could lead to incorrect account closure reports.

Metadata (including lineage): Metadata is the foundation of quality output. It helps characterize data and helps your organization understand and consume it easily. Metadata should explain the who, what, when, how and why of data — it can even provide information on things like the ownership of data product code.

Preparing a test strategy: Quality starts with a conversation

Our experience suggests that when data producer teams take ownership of the data testing process, data quality is more easily and consistently maintained. But a robust test strategy requires collaboration between data consumers and data producers. Data consumers need to address the below areas to develop a data test strategy:

- What quality features are important – is it completeness, distinctness, compliance or something else?
- What business requirements are we building?
- What target value should producers aim and optimize for?
- Identifying domain-driven quality metrics – for example, the needs of retail would be quite different to the needs to real estate

Data producer teams should also aim to capture finer-grained metrics such as:

- Error tolerance
- Ownership – if a quality check is broken who is going to fix it and when?
- How much is data quality worth? Will adding more data improve your analytics? What are the appropriate and agreed thresholds and tolerances for data quality for the business? Is 90%, 99%, 99.9% accuracy expected or acceptable for the end user of the data?
- Service level agreements (SLAs) – how much downtime does the business allow for each year?

As the owners of data quality, data producers are ultimately responsible for knowing these thresholds and meeting agreed expectations.

Implementing the strategy

With a solid test strategy in place, the next thing to consider is implementing data quality tests as a write-audit-publish (WAP) pattern. Using this pattern, you write data and audit the results before you publish them. That will allow you to make corrections before publishing.

Enabling new data ingestion within continuous integration and continuous integration/continuous delivery (CI/CD) pipelines also ensures imported data goes through quality tests – and doesn't break existing checks. There may be instances where checks should break the pipeline and send a high urgency alert. If a check flags a negative house number within a pipeline running on real estate data for instance, you need to immediately address the issue and stop the run. Whereas if a house number is simply out of range, you can continue the pipeline runs with simple alerts.

Making the results from these quality checks available to the wider business is incredibly important. For example, an address list that is 15% incomplete may delay the marketing team's campaign launch. While a variance of 1% in an engineering measurement could jeopardise an expensive manufacturing process. Making quality levels visible as part of a metadata catalog can also be immensely valuable, as it allows data consumers to make informed decisions when considering the data's use cases.

Making quality levels visible as part of a metadata catalog can also be immensely valuable, as it allows data consumers to make informed decisions when considering the data's use cases.

Many data quality frameworks today offer profiling reports that include the error/failure distribution. You can find some good open-source frameworks – we've had positive experiences with Great Expectations, Deequ and Soda – that can help you implement data quality tests through a range of features. Depending on the level of integration you require, some key features to consider for your framework include:

- Using an open source solution to avoid vendor lock-in
- How results can be visualized and shared to ensure transparency across the organization
- Implementing data validation on incremental loads to ensure checks are performed on an ongoing basis with whatever is the desired ingest frequency
- Implementing anomaly detection to automatically catch and raise alerts for unexpected deviations above or below a certain tolerance
- Integration with alerting and monitoring tools to ensure visibility on the system without building observability integrations yourself (and speeden the time to roll out the checks)
- Integrating with the data catalog to avoid building discoverability integrations yourself and create visibility from the start
- Programming language support — choosing a framework based on the technical stack of your current data ecosystem

Just enough testing at just the right time

You should consider implementing data checks at various stages - ensuring quality is maintained throughout the Extract, Transform, and Load (ETL) pipelines and standalone tests for complex data transformations. Think about this as shifting data quality focus to the left – or having checks as early as possible. Implement schema and basic integrity checks during raw ingestion itself, **as well as** during the transformation stage. Organizational data quality improves by building different layers of tests as you pass through the data pipeline.

Treat your development environment and pipeline tests just as you would treat them in a production setting. Communicate with source application teams about data quality issues and fix them at the source application. For example, Know Your Customer (KYC) checks in your pipeline need to have non-nullable customer attributes. If the source application doesn't enforce a validation on the system, null/empty values will be ingested – making the transformations pointless or invalid for many rows of data. Monitoring metrics such as row counts, totals and averages and setting timely failure alerts will also reduce time-to-resolution.

Build trust and save time through data quality

A robust data quality test suite with a focus on trust, relevance and repeatability will go a long way to instill confidence in your data consumers and reduce the amount of time spent on quality issues.

Organizations need to build infrastructure that enables data producer teams to fix and resolve issues and deploy changes quickly to continually maintain and evolve the data quality paradigm. Once they do that, they can begin focusing more on value-adding activities, rather than simply fixing problems.

Measuring and communicating data quality will help you achieve alignment on the state and business relevance of data across the organization. It will also allow data teams to make continuous improvements over time.

Shift left on security and privacy: why it's critical to speed, quality and customer trust

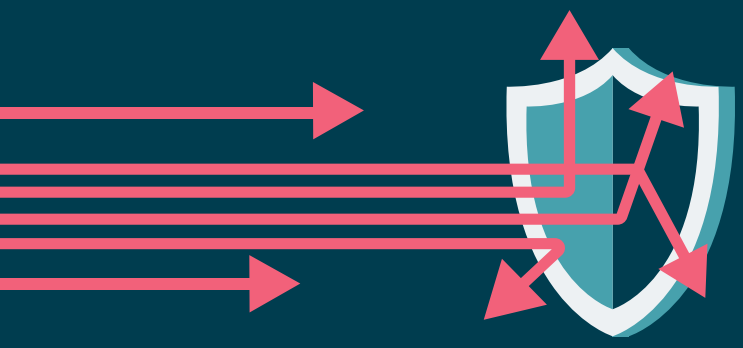
By Mitchell Lisle and Harmeet Kaur Sokhi

In the first half of 2022, there were **817 data compromises impacting over 53 million people** in the United States. Each of these security breaches cost an **average $4.35 million** (US)– a 12.7% increase since 2020.

Governments around the world are implementing stricter laws around data privacy, as more organizations and individuals are affected by breaches every day. As engineering teams play an increasingly important role in this space, it wouldn't be right to wrap up our deep dive into data engineering without discussing security and privacy.

Security and privacy are often used interchangeably, but they are not the same. Security enables privacy, but doesn't guarantee it. **Privacy** typically refers to a user's ability to control, access, and regulate their personal information, while **security** refers to the system that protects that data from getting into the wrong hands. You can have data security without data privacy, but not the other way around.

They are equally important and any good information management system will ensure personal data is treated appropriately.



Privacy typically refers to a user's ability to control, access, and regulate their personal information, while security refers to the system that protects that data from getting into the wrong hands.

Why shift left?

Too often security and privacy is compromised early on in development projects simply by being overlooked. While this might mean you can initially move fast, over time you'll need to invest significant time and energy refactoring their software for security and privacy.

To make things even more complicated, the challenges of doing this to a product or solution that is already in production can lead to further risks as it increases the surface area for security or data breaches. For any product that processes data that could be considered Personally Identifiable Information (PII), security and privacy is especially critical to consider from the very beginning of a project.

And that is what we mean by "shifting left:" In software engineering, shifting left is a conscious effort to embed certain practices earlier in the development lifecycle – left being the start of a product lifecycle, right being the end.

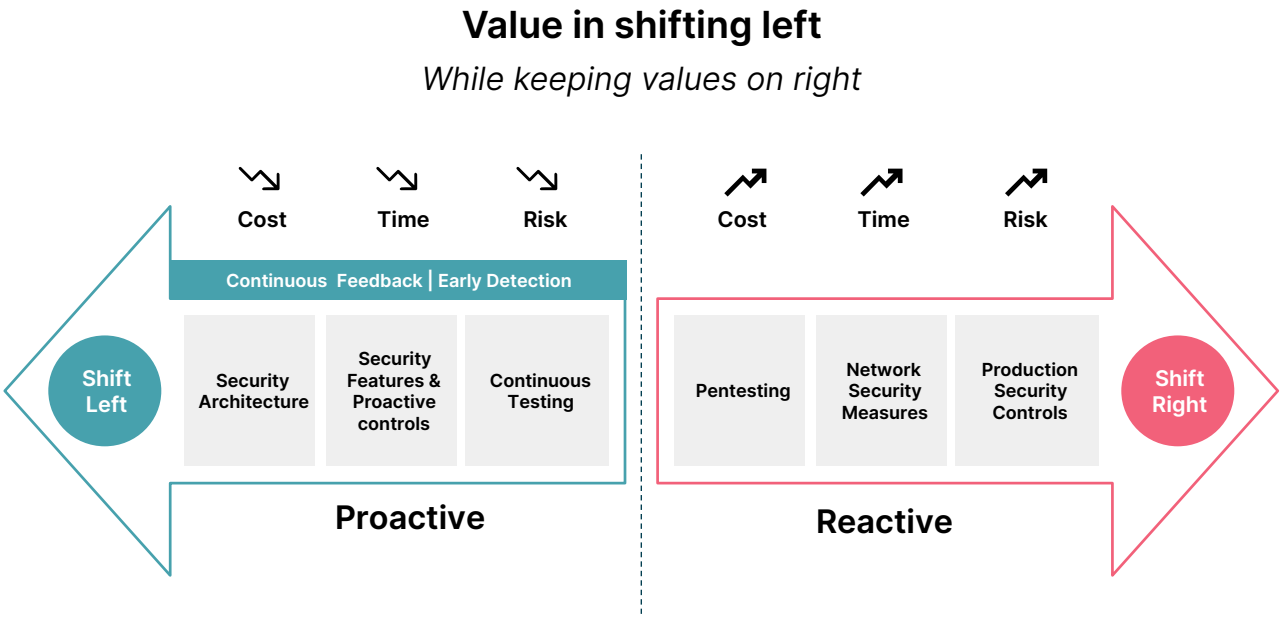


Figure 1: Value in shifting left

Many organizations will have security engineers or even a Chief Information Security Officer (CISO), yet many lack technical expertise when it comes to privacy. This has led to the emergence of the **privacy engineer** – a specialist software engineering role that ensures privacy considerations are embedded into product development, rather than left as an afterthought. The need for this role has intensified partly because organizations there are today increasing legislative requirements to which practices, processes and products must comply. Greater awareness of the ethical dimension of technology also makes the role particularly valuable. In the past, data collection was something often done with little consideration for users' personal privacy. However, today privacy can be a differentiator: there are **plenty of examples** of privacy being placed at the center of a product development.

In the same way that we — as developers — think about technical debt, we need to also start paying attention to our **"privacy debt"**. With data breaches increasing, companies have a decision to make about when they tackle this debt. Those that successfully shift left on security and privacy will significantly reduce the probability of ending up on a **growing list of companies who have failed to protect their data**.

Safeguarding data

Personally Identifiable Information (PII) is data that directly identifies, in isolation or in combination with other data, an individual. Without the right security in place, hackers can access this data and create profiles – using that information to impersonate them or sell it to other criminals.

Consideration is required when storing any form of personal data. Stop and reflect: is collecting PII necessary to deliver the customer experience? Can you retain trust? For example, a retail recommendation system can offer a tailored experience with a broad age bracket without capturing a customer's date of birth. For organizations that do use PII, being able to find and identify sensitive data is critical to protecting their customers and their reputation. Technologies like data catalogs and appropriate governance frameworks are particularly useful here for ensuring data can be effectively organized and secured.

It is important to note that simply obfuscating or masking PII fields in a dataset does not necessarily de-identify an individual's data. It may be possible to re-identify the data **using other contextual information**. For example, hackers use uniqueness as a path to exploit vulnerabilities, so knowing all the ways your data can be unique to an individual is important. A driver of a distinctly coloured car in a large city might be fairly unique, but that same driver in a small country town would be identified easily. This also goes for machine learning approaches that are trained on data with outliers. Outliers can reveal sensitive information about your data and can inadvertently leak it through a prediction API.

One of the complexities organizations face is the need for access to PII to allow development teams to experiment and test as they're working. Test environments are often not subject to the same security and privacy, because they don't contain production data. But a data science workflow needs to have data that represents production, so you can train models and do analysis to understand what that model will do in production.

There are many ways you can do this without giving access to production data directly:

1. Generate fake data that matches the schema of your production data. For some use cases, such as data validation checks, this can be enough to help ensure pipelines work adequately and without error. But if your goal is to train and release a model into production, you should avoid fake data. Not training your model on data that is as close to the real thing as possible raises ethical and accuracy concerns.
2. Evaluate whether **synthetic data**, or data that is representative of the real scenario but generated by a model, could work for your use case. You may still need to apply additional privacy preserving techniques on top of this data.
3. Generate a subset of secure, anonymous data. Anonymization is a challenging and at times impossible task when it comes to PII. Simply removing obvious fields like names, addresses and other identifiers does not mean you cannot re-identify an individual in that dataset. Having a good understanding of privacy engineering practices such as masking, differential privacy and encrypted computation are important to do this effectively.
4. Build an isolated secure environment specifically for model building and training with access to a copy of production data. This approach is more costly and introduces risk since you're copying data to another location. You'll also need a separate environment with all of the same security and privacy controls as production.

The biggest shift you have to make to improve security and privacy is to think small. **Data minimization** is your friend – helping you build what you want, using the smallest subset of data you really need.

Adopt sensible security practices

There are a number of practices you can put in place to help you make decisions when it comes to developing secure data products.

1. **Appointing security champions** with experience in security activities and processes will help guide your development teams on the right decisions to make.
2. Put a **data classification process** in place to allow you to tag sensitive data and apply governance policies across the organisation based on the sensitivity of your data.

   Here is a simple mental model for your data as it enters your systems.

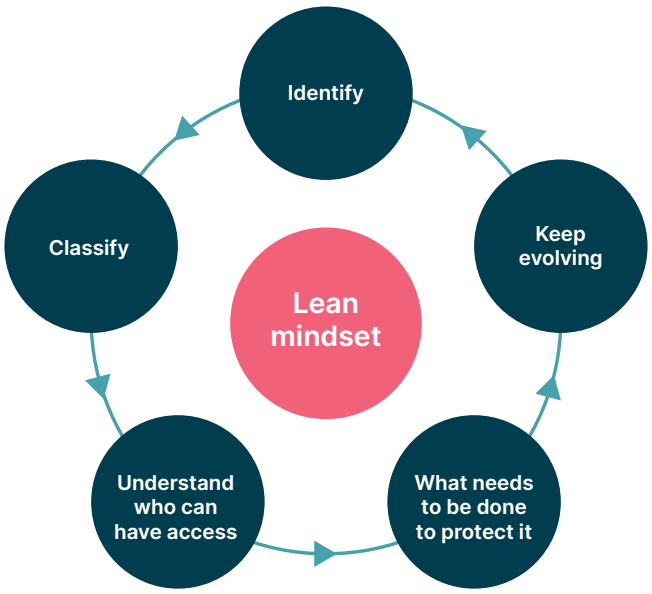


Figure 2: Lean mindset

3. Run frequent **security workshops**, such as threat modelling, to so you can make impact incrementally, prioritizing work as you go. Focus on small, actionable changes you can make to ensure these sessions continue to deliver value to your security infrastructure.
4. Use the **CIA** (Confidentiality, Integrity, Availability) **triad** to think about security:
   - **Confidentiality:** The asset cannot be accessed by people or systems that shouldn't access it.
   - **Integrity:** The asset cannot be changed by people or systems that shouldn't change it.
   - **Availability:** Every person and system that should be able to access an asset can do so.

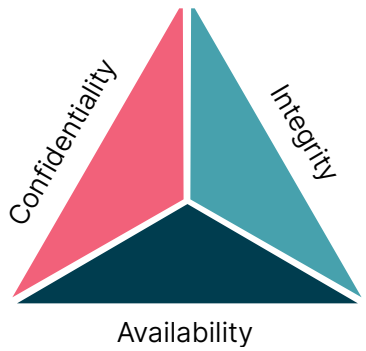


Figure 3: CIA triad

Here are some security considerations at each phase of a project lifecycle:

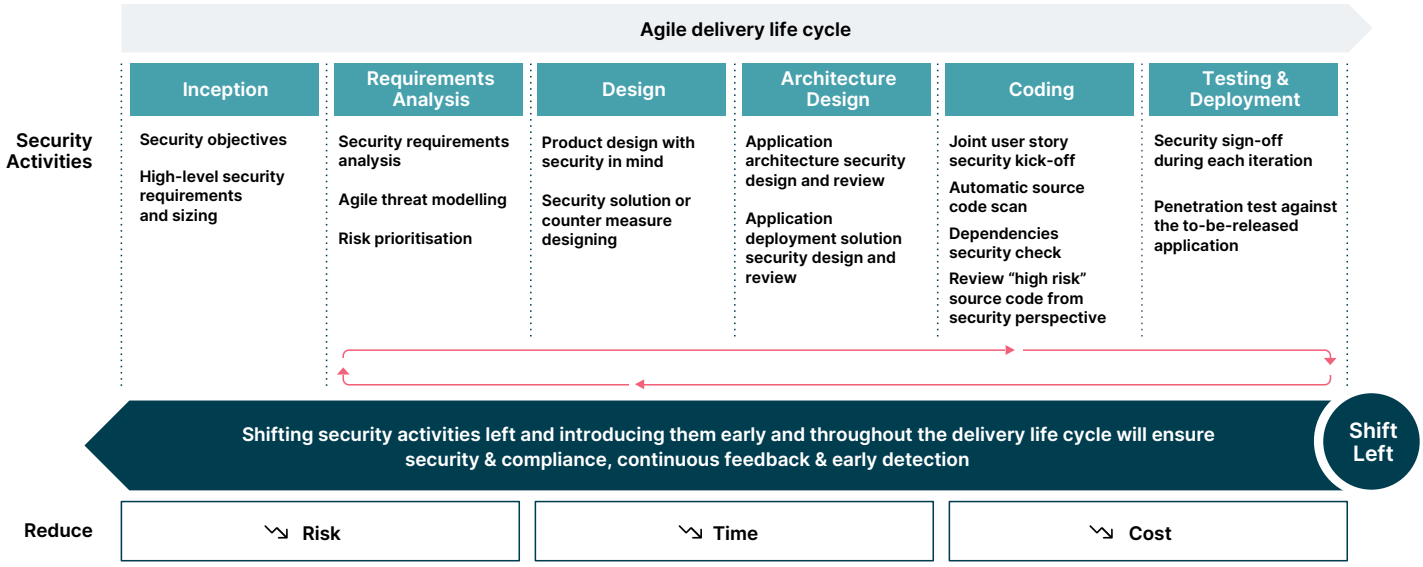


Figure 4: Phases of a project lifecycle

Read our Responsible tech playbook for more useful tools, techniques and principles. Use it to inform your next planning session, and to better address critical security and privacy considerations; many of the tools and frameworks you find in it will help you to assess your own risks.

Invest in security from the start

Building security and privacy into your process will improve the quality of your software and enable you to move faster without having to rely on major refactors to ensure your systems are up to standard.

The later you leave security and privacy in the development process, the more at risk you are of a significant data breach or security incident. Investing in everything needed to embed security and privacy in your system is well worth it in the long run: for your organization and your customers.

Ready to unlock data's full potential?

Data's potential is indisputable. But in a fast-paced, ever-evolving landscape, you need the right data engineering practices to leverage it to give your organization a competitive advantage.

Modern data engineering helps you get the most value from your data, make better decisions and create more tailor customer experiences – at speed.

But to truly harness modern practices, your business needs to embrace a new way of thinking about data. Shifting to a data product mindset requires an organization-wide cultural change. You may also need to realign internal structures and platforms to empower data teams.

Because an empowered team with the right set of skills, experience and knowledge will be able to develop the right solutions for the right problems faster – and thrive.

And by applying sensible defaults to your practices and principles, your data teams can add more value, reduce project risks, and find opportunities for improvement while delivering data products at speed.

Shifting to modern data engineering practices can be complex and take time. At Thoughtworks, we have helped many organizations make the move – and unlock the true potential of their data at scale. If you need help with any of the areas discussed in this playbook or would like to share your experience with modern data engineering, get in touch now.

solutions@thoughtworks.com
thoughtworks.com/en-au/what-we-do/data-and-ai

About Thoughtworks

Thoughtworks is a global technology consultancy that integrates strategy, design and engineering to drive digital innovation. We are over 12,500 people strong across 50 offices in 18 countries. Over the last 25+ years, we've delivered extraordinary impact together with our clients by helping them solve complex business problems with technology as the differentiator.

thoughtworks.com

Get in touch with us

Thoughtworks has offices in:

Gadigal Country / Sydney
Land of the Gadigal People of the Eora Nation
Level 10, 50 Carrington Street
Sydney, New South Wales 2000
Phone +61 2 9224 1700

Meeanjin / Brisbane
Land of the Turrbal and Jagera/Yuggera Peoples
Level 19, 127 Creek Street
Brisbane, Queensland 4000
Phone +61 7 3129 4506

Naarm / Melbourne
Land of the Wurundjeri People of the Kulin Nation
Level 23, 303 Collins Street
Melbourne, Victoria 3000
Phone +61 3 9691 6500

/thoughtworks