O'REILLY®

Data Mesh

Delivering Data-Driven Value at Scale

Early Release
Raw & Unedited

Compliments of

Zhamak Dehghani

Data Mesh

Delivering Data-Driven Value at Scale

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Zhamak Dehghani

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Data Mesh
by Zhamak Dehghani

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Melissa Duffield
Development Editor: Gary O'Brien
Production Editor: Beth Kelly
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

January 2022: First Edition

Revision History for the Early Release

2021-06-18: First Release
2021-07-28: Second Release
2021-09-07: Third Release
2021-10-07: Fourth Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492092391* for release details.

978-1-492-09232-2

[FILL IN]

Table of Contents

PART I
Why Data Mesh?

By doubting we are led to question, by questioning we arrive at the truth.
—Peter Abelard

Data Mesh is a new approach in sourcing, managing, and accessing data for analytical use cases at *scale*. Let's call this class of data, analytical data. Analytical data is used for predictive or diagnostic use cases. It is the foundation for visualizations and reports that provide insights into the business. It is used to train machine learning models that augment the business with data-driven intelligence. It is the essential ingredient for organizations to move from intuition and gut-driven decision-making to taking actions based on observations and data-driven predictions. Analytical data is what powers the software and technology of the future. It enables a technology shift from human-designed rule-based algorithms to data-driven machine-learned models. Analytical data is becoming an increasingly critical component of the technology landscape.

The phrase data in this writeup, if not qualified, refers to analytical data. Analytical data serves reporting and machine learning training use cases.

Data Mesh calls for a fundamental shift in our assumptions, architecture, technical solutions, and social structure of our organizations, in how we manage, use, and own analytical data.

- *Organizationally*, it shifts from centralized ownership of the data by specialists who run the data platform technologies, to a *decentralized data ownership model* pushing ownership and accountability of the data back to the business domains where it originates from or is used.
- *Architecturally*, it shifts from collecting data into monolithic warehouses and lakes to connecting data through a *distributed mesh* of data accessed through standardized protocols.
- *Technologically*, it shifts from technology solutions that treat data as a by-product of running pipeline code, to solutions that treat data and code that maintains it as one lively autonomous unit.
- *Operationally*, it shifts data governance from a top-down centralized operational model with human interventions, to a *federated model* with computational policies embedded in the nodes on the mesh.
- *Principally*, it shifts our value system from data as an asset to be collected, to *data as a product* to serve and delight the users.

Figure I-1 summarizes the dimensions of shift that Data Mesh introduces.

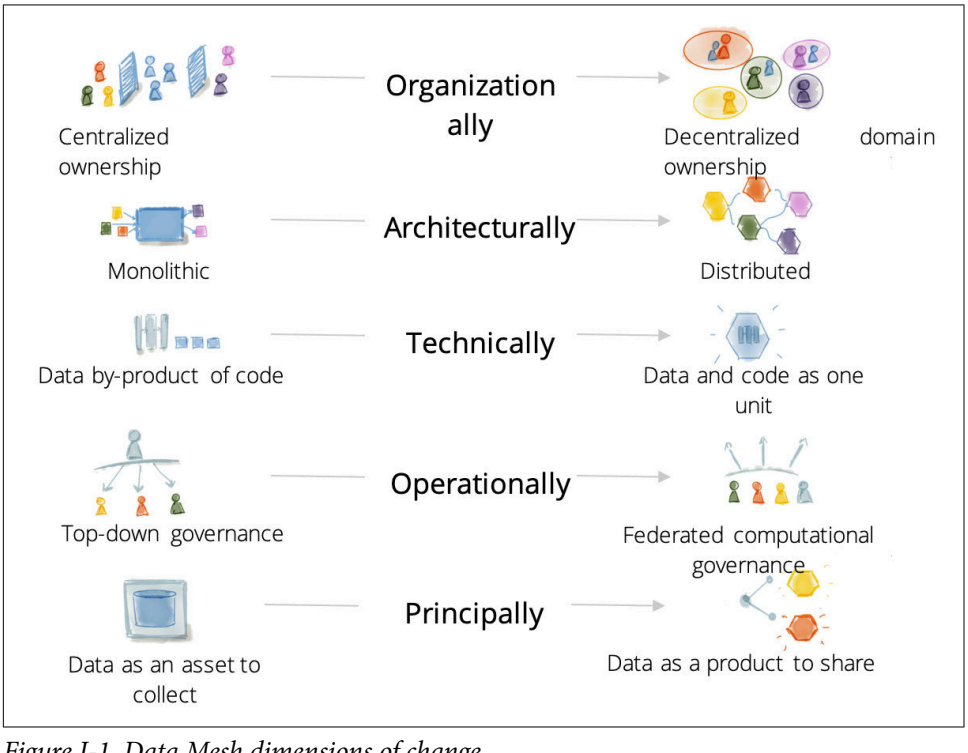

Figure I-1. Data Mesh dimensions of change

This is quite a shift, and an uncomfortable one. So why do we need it, and why now? I will answer this question in part I of the book.

In the first chapter we look at the macro drivers, the current realities that have pushed us to a tipping point, where our past evolutionary approaches no longer serve us. The second chapter introduces the core outcomes that Data Mesh achieves through its shifts in approach. And in the last chapter of part I, we briefly review the history of analytical data management architecture and why what got us here will no longer take us to the future.

Let's set the stage for Data Mesh.

CHAPTER 1
The Inflection Point

A note for Early Release readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

> *A strategic inflection point is a time in the life of business when its fundamentals are about to change. That change can mean an opportunity to rise to new heights. But it may just as likely signal the beginning of the end.*
>
> —Andrew S. Grove, CEO of Intel Corporation

Data Mesh is what comes after an inflection point, shifting our approach, attitude, and technology toward data. Mathematically, an inflection point is a magic moment at which a curve stops bending one way and starts curving in the other direction. It's a point that the old picture dissolves, giving way to a new one.

This won't be the first or the last inflection point in the evolution of data management. However, it is the one that is most relevant now. There are drivers and empirical signals that point us in a new direction. I personally found myself at this turning point in 2018. When many of our clients at ThoughtWorks, a global technology consultancy, simultaneously were seeking for a new data architecture that could respond to the scale, complexity, and aspirations of their business. After reading this chapter, I hope that you too arrive at this critical point, where you feel the urge for change, to

wash away some of the fundamental assumptions made about data, and imagine something new.

Figure 1-1 is a simplistic demonstration of the inflection point in question. The x-axis represents the macro drivers that have pushed us to this inflection point. They include an ever-increasing business complexity combined with uncertainty, proliferation of data expectations and use cases, and the availability of data from ubiquitous sources. On the y-axis we see the impact of these drivers on business agility, ability to get value from data and resilience to change. In the center is the inflection point, where we have a choice to make. To continue with our existing approach and, at best, reach a plateau of impact, or take the Data Mesh approach with the promise of reaching new heights in the agility of acting on data, immunity to rapid change, and being able to get value from data at a larger scale. Part II of this book will go through the details of what the Data Mesh approach entails.

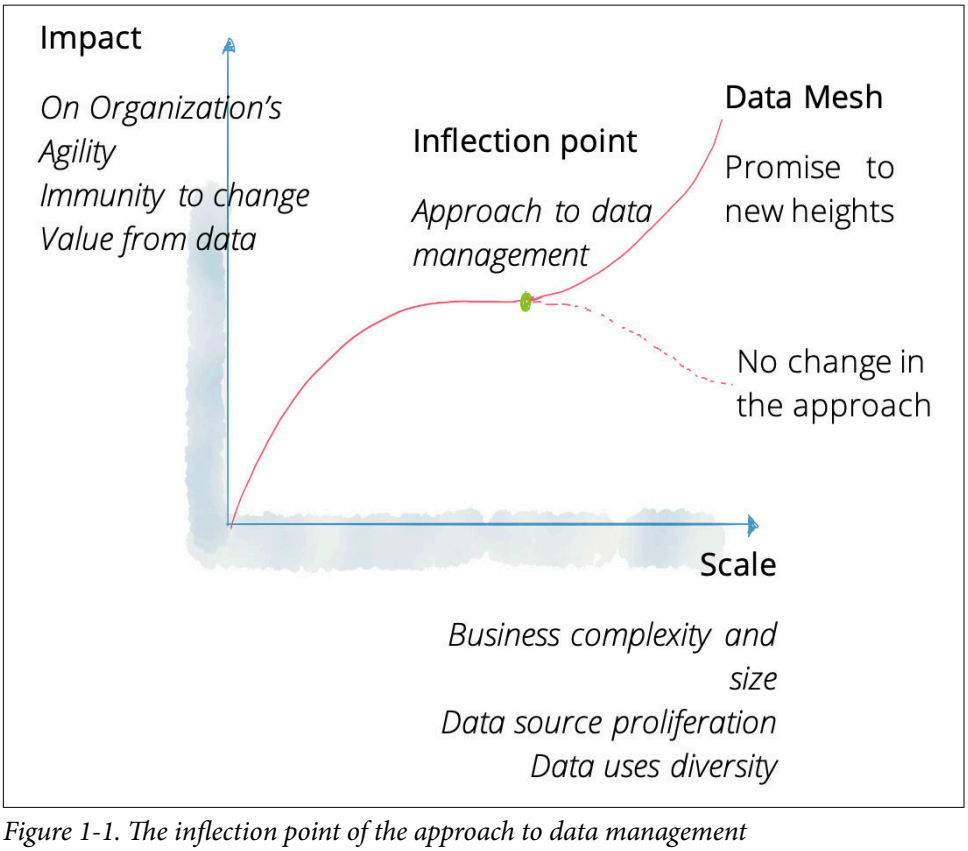

Figure 1-1. The inflection point of the approach to data management

In this chapter, I share today's data landscape realities that are the main drivers for Data Mesh.

Great Expectations of Data

One of the perks of being a technology consultant is traveling through many industries and companies, and getting to know their deepest desires and challenges. Through this journey, one thing is evident: being a data-driven organization remains one of the top strategic goals of executives.

Here are a few examples, all truly inspiring:

> *Our mission at Intuit is to power prosperity around the world as an AI-driven expert platform company, by addressing the most pressing financial challenges facing our consumer, small business and self-employed customers.*
>
> —Financial SaaS Company

> Our mission is to improve every single member's experience at every single touchpoint with our organization through data and AI.
>
> —Healthcare provider and payer company

> By People, For People: We incorporate human oversight into AI. With people at the core, AI can enhance the workforce, expand capability and benefit society as a whole.
>
> —Telco

No matter the industry or the company, it's loud and clear, we want to become intelligently empowered[1] to:

- provide the best customer experience based on data and hyper-personalization
- reduce operational costs and time through data-driven optimizations
- empower employees to make better decisions with trend analysis and business intelligence

All of these scenarios require data--a high volume of diverse, up-to-date, and truthful data that can, in turn, fuel the underlying analytics and machine learning models.

A decade ago, many companies' data aspirations were mainly limited to business intelligence (BI). They wanted the ability to generate reports and dashboards to manage operational risk, respond to compliance, and ultimately make business decisions based on the facts, on a slower cadence. In addition to BI, classical statistical learning has been used in pockets of business operations in the industries such as insurance, healthcare, and finance. These early use cases, delivered highly specialized teams, have been the most influential drivers for many past data management approaches.

1 Christoph Windheuser, What is Intelligent Empowerment?, (ThoughtWorks, 2018).

Today, data aspirations have evolved beyond business intelligence to every aspect of an organization, using machine learning in the design of the products, such as automated assistants, in the design of our services and experience of our customers, such as personalized healthcare, and streamlining operations such as optimized real-time logistics. Not only that, the expectation is to democratize data, so that the majority of the workforce can put data into action.

Meeting these expectations requires a new approach to data management. An approach that can seamlessly fulfill the *diversity of modes of access to data*. Access that ranges from a simple structured view of the data for reporting, to a continuously reshaping semi-structured data for machine learning training; from real-time fine-grained access to events to aggregations. We need to meet these expectations with an approach and architecture that natively supports diverse use cases and does not require copying data from one technology stack to another across the organization so that we can meet the needs of yet another use case.

More importantly, the widespread use of machine learning requires a new attitude toward application development and data. Need to move from deterministic and rule-based applications - where given a specific input data, the output can be determined - to nondeterministic and probabilistic data-driven applications - where given a specific input data, the output could be a range of possibilities which can change over time. This approach to application development requires continuous refining of the model over time, and continuous, frictionless *access to the latest data*.

The great and diverse expectations of data require us to step back, acknowledge the accidental technical complexities that we have created over time, and wonder if there is a simpler approach to data management that can universally address the diversity of needs today, and beyond.

The Great Divide of Data

Many of the technical complexities organizations face today stem from how we have divided the data -- *operational* and *analytical data*, siloed the teams that manage them, proliferated the technology stacks that support them and how we have integrated them.

Today, we have divided the data and its supporting technology stacks and architecture into two major categories: *operational data*: databases that support running the business and keeping the current state of the business - also known as transactional data; and *analytical data*: data warehouse or lake providing a historical, integrated and aggregate view of data created as a byproduct of running the business. Today, operational data is collected and transformed to form the analytical data. Analytical data trains the machine learning models that then make their way into the operational systems as intelligent services.

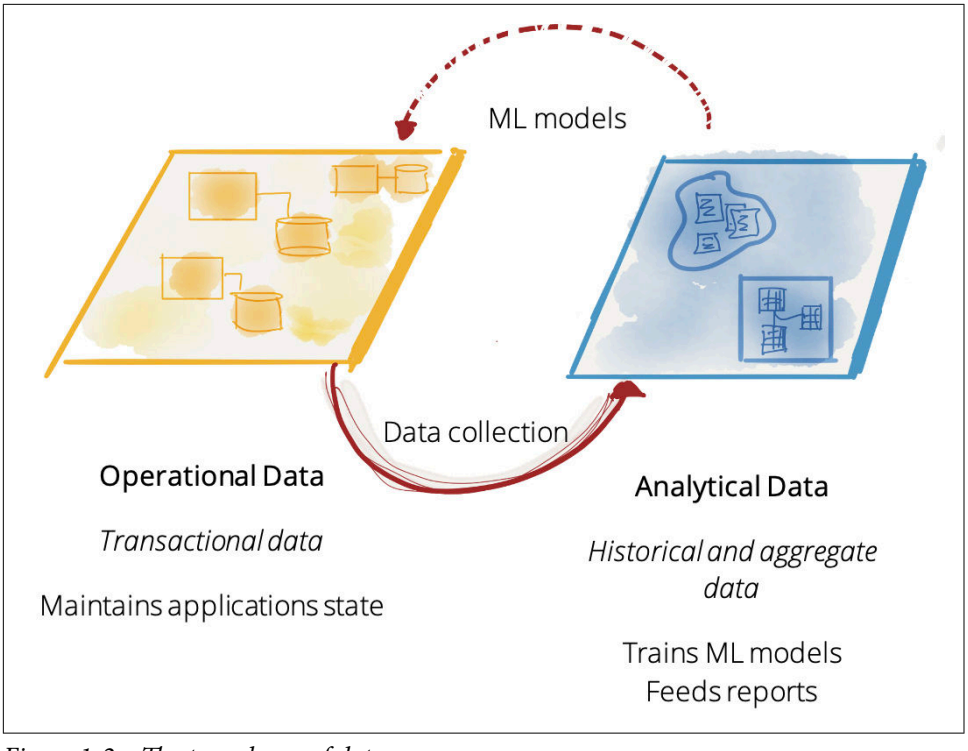

Figure 1-2. : The two planes of data

Operational Data

Operational data sits in databases of microservices, applications, or systems of records that support the business capabilities. Operational data keeps the current state of the business. It is optimized for application's or microservice's logic and access patterns. It often has a transactional nature. It's referred to as data on the inside, private data of an application or a microservice that performs CRUD (create, update, delete) operations on it. Operational data is constantly updated, so it's access requires reads and writes. The design has to account for multiple people updating the same data at the same time in unpredictable sequences (hence the need for transactions). The access is also about relatively in-the-moment activity. Operational data is recording what happens in the business, supporting decisions that are specific to the business transaction. In short, *operational data is used to run the business and serve the users.*

Imagine a digital media streaming business that streams music, podcast and other digital content to its subscribers and listeners. Registration service implements the business function of registering new users or unregistering them. The database that

supports the registration and deregistration process, keeping the list of users, is considered operational data.

INTRODUCING DAFF INC.

Daff Inc. is a global digital streaming company, that started its journey with streaming music and rapidly growing to provide other digital audio experiences such as sharing podcasts, radio shows and audio books. Daff Inc. serves both paid and free subscribers. Daff's mission is to get everyone's audio content heard; from independent artists, podcasters, to record labels and larger publishers. Daff hosts social events to connect the audience with the performers.

Daff Inc. is making big investments in a robust data and AI infrastructure to become data-driven; use their data to optimize every single aspect of their business. Serve the listeners with recommendations specialized to their taste, mood, time of day and location; empower the artists with information about their listeners such as locations, listeners profile, campaign results to help them refine their work; optimize the quality of their digital services using users and digital players captured events; and ultimately streamline their business operations such as artist onboarding, payments and advertisements using up to date and accurate data.

The word 'Daff' is the name of a Persian percussion instrument, dated more than 3000 years.

Analytical Data

Analytical data is the temporal, historic and often aggregated view of the facts of the business over time. It is modeled to provide retrospective or future-perspective insights. Analytical data is optimized for analytical logic - training machine learning models, creating reports and visualizations. Analytical data is called data on the outside, data directly accessed by analytical consumers. Analytical data is immutable and has a sense of history. Analytical use cases require looking for comparisons and trends over time, while a lot of operational uses don't require much history. The original definition of analytical data as *a nonvolatile, integrated, time variant collection of data*[2] still remains valid.

In short, analytical data is used to optimize the business and user experience. This is the data that fuels the AI and analytics aspirations that we talked about in the previous section.

For example, in the case of Daff Inc. it's important to optimize the listeners' experience with playlists recommended based on their music taste and favorite artists. The

2 Definition provided by William H. Inmon known as the father of data warehousing.

analytical data that helps train the playlist recommendation machine learning model, captures all the past behavior of the listener as well as all characteristics of the music the listener has favored. This aggregated and historical view is analytical data.

Over time, the analytical data plane itself has diverged into two generations of architectures and technology stacks: initially data warehouse and followed by data lake; with data lake supporting data science access patterns and preserving data in its original form, and data warehouse supporting analytical and business intelligence reporting access patterns with data conforming to a centrally unified ontology. For this conversation, I put aside the dance between the two technology stacks: data warehouse attempting to onboard data science workflows and data lake attempting to serve data analysts and business intelligence.

Analytical and Operational Data Misintegration

The current state of technology, architecture and organization design is reflective of the divergence of the analytical and operational data planes - two levels of existence, integrated yet separate. Each plane operates under a different organizational vertical. Business intelligence, data analytics and data science teams, under the leadership of Chief Data and Analytics officer (CDAO), manage the analytical data plane, while business units and their corresponding technology domains manage the operational data. From the technology perspective, there are two independent technology stacks that have grown to serve each plane, while there are some convergence such as infinite event logs.

This divergence has led to the two-plane data topology and a fragile integration architecture between the two. The operational data plane feeds the analytical data plane through a set of scripts or automated processes often referred to as ETL jobs - Extract, Transform, and Load. Often operational databases have no explicitly defined contract with the ETL pipelines for sharing their data. This leads to fragile ETL jobs where unanticipated upstream changes to the operational system and their data leads to downstream pipeline failures. Over time the ETL pipelines grow in complexity trying to provide various transformations over the operational data, flowing data from the operational data plane to the analytical plane, and back to the operational plane.

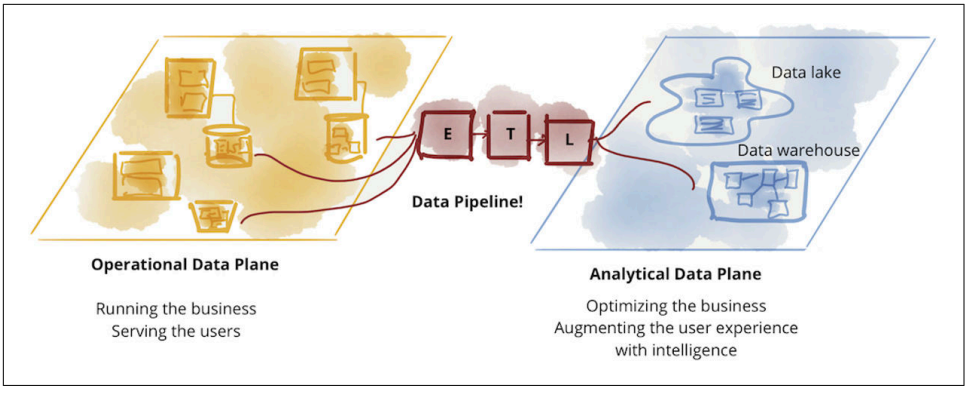

Figure 1-3. Pipeline-based integration of the data planes

The challenges of the two-plane data management approach with a brittle integration through pipelines, and a centralized data warehouse or lake for access to data is a major driver to reimagine the future solutions.

Scale, Encounter of a New Kind

Since the mid 2000s, we have evolved our technologies to deal with the scale of the data in terms of its *volume, velocity and variety*. We built the first generation batch data processing to manage the large volume of data that our applications and touchpoints generated, we built stream processing architectures to handle the speed of data that started flowing from our mobile devices, and built different types of storage systems to manage the diversity of data, text, imaging, voice, graphs, files, etc. Then we got carried away and kept tagging more Vs to data to encourage access to clean data - *veracity* - and aim to get *value*[3] from data.

Today, we are encountering a new kind of scale, the *origins and location of the data*. The data-driven solutions often require access to data beyond a business domain, organizational or technical boundary. The data can be originated from every system that runs the business, from every touchpoint with customers, and from other organizations. The next approach to data management needs to recognize the proliferation of the origins of the data, and their ubiquitous nature.

The most interesting and unexpected patterns emerge when we connect data from a variety of sources, when we can have access to information that is beyond the transactional data that we generate running our business. The future of intelligent healthcare requires a longitudinal human record of a patient's diagnostics, pharmaceutical records, personal habits, etc. and in comparison with all other patients' history. These

3 https://www.bbva.com/en/five-vs-big-data

sources are beyond a single organization's control. The future of intelligent banking requires data beyond the financial transactions that customers perform with their banks. They'll need to know the customers' housing needs, the housing market, their shopping habits, their dreams, to offer them the services they need, when they need it.

This unprecedented scale of diversity of sources, requires a shift in data management. A shift away from collecting data from sources into one big centralized place, repeatedly across every single organization, to *connecting data*, wherever it is.

Beyond Order

I'm writing this book during the pandemic of 2020-2021. If there was any doubt that our organizations need to navigate *complexity, uncertainty and volatility*, the pandemic has made that abundantly clear. Even on a good day outside of the pandemic, the complexity of our organizations demand a new kind of immunity, *immunity to change*.

The complexity that has risen from the ever changing landscape of a business is also reflected in the data. Rapid delivery of new features to products, new and changed offerings and business functions, new touchpoints, new partnerships, new acquisitions, all result in a continuous reshaping of the data.

More than ever now, organizations need to have the pulse of their data and the ability to act quickly and respond to change with *agility*.

What does this mean for the approach to data management? It requires access to the quality and trustworthy facts of the business at the time they happen. The data platforms must *close the distance* - time and space - between when an event happens, and when it gets consumed and processed for analysis. The analytics solutions must guide *real time decision making*. Rapid response to change is no longer a premature optimization[4] of the business; it's a baseline functionality.

Data management of the future must build-in change, by default. Rigid data modeling and querying languages that expect to put the system in a straitjacket of a never-changing schema can only result in a fragile and unusable analytics system.

Data management of the future must embrace the complex nature of today's organizations and allow for *autonomy* of teams with *peer-to-peer* data collaborations.

Today, the complexity has stretched beyond the processes and products to the technology platforms themselves. In any organization, the solutions span across multiple cloud and on-prem platforms. The data management of the future must support

4 Donald Knuth made the statement, "(code) premature optimization is the root of all evil."

managing and accessing data *across multiple cloud providers*, and on-prem data centers, by default.

Approaching the Plateau of Return

In addition to the seismic shifts listed above, there are other telling tales about the mismatch between data and AI investment and the results. To get a glimpse of this, I suggest you browse the NewVantage Partners annual reports; an annual survey of senior corporate c-executives on the topics of data and AI business adoption. What you find is the recurring theme of an increasing effort and investment in building the enabling data and analytics platforms, and yet experiencing low success rates.

For example, in their 2021 report, only 26.8% of firms reported *having forged a data culture.* Only 37.8% of firms reported that they have *become data-driven,* and only *45.1%* of the firms reported that they are *competing using data and analytics.* It's too little result for the pace and amount of investment; 64.8% of surveyed companies reported greater than $50MM investment in their Big Data and AI strategies.

Despite continuous effort and investment in one generation of data and analytics platforms to the next, the organizations find the results middling.

I recognize that the organizations face a multi-faceted challenge in transforming to become data-driven; migrating from decades of legacy systems, resistance of a legacy culture to rely on data, and competing business priorities.

The future approach to data management must look carefully at this phenomena, why the solutions of the past are not producing a comparable result to the human and financial investment we are putting in today. Some of the root causes include lack of skill sets needed to build and run data and AI solutions, organizational, technology and governance bottlenecks, friction in discovering, trusting, accessing and using data.

Recap

As a decentralized approach to managing data, Data Mesh embraces the data realities of organizations today, and their trajectory, while acknowledging the limitations our solutions face today.

Data Mesh assumes a new default starting state: proliferation of data origins within and beyond organizations boundaries, on one or across multiple cloud platforms. It assumes a diverse range of use cases for analytical data from hypothesis-driven machine learning model development to reports and analytics. It works with a highly complex and volatile organizational environment and not against it.

In the next two chapters, I set the stage further. Next, we look at our expectations from Data Mesh as a post-inflection-point solution. What organizational impact we expect to see, and how Data Mesh achieves them.

CHAPTER 2
After The Inflection Point

A note for Early Release readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

> *The only way to make sense out of change is to plunge into it, move with it, and join the dance.*
>
> —Alan Watts

Standing at an inflection point is a magical experience. It's where we learn and depart from the past and choose a new path. It's a point where we have a choice to turn to a new direction. The rest of this book provides instructions on how to move toward this new direction with Data Mesh. We will discuss what constitutes Data Mesh in Part II, how to architect it in Part III, and how to begin executing it in Part IV. However, before we dive into that, in this chapter, I'd like to introduce Data Mesh based on its *impact* on organizations, given the *environmental conditions* it operates in, and the *issues from the past solutions* it must address.

Data Mesh must accept the *environmental conditions* that we discussed in Chapter 1, as the default starting point. It must assume, by default, the *ubiquitous nature of the data*. Data can be of any origin, it can come from systems within an organization, or outside, beyond the boundary of organizational trust. It can be physically served by

any underlying platform on one cloud hosting or another. Data Mesh must embrace the *diversity of the use cases* and their unique modes of access to data. The use cases range from historical data analysis and reporting, training machine learning models and data-intensive applications. Each needs to read data in a different format, in the spectrum of graphs, files, tables and events. An ever increasing *complexity of the business* landscape--its diversity of functions, continuous change, and need for real-time decision making in volatile times--is the organizational reality within which Data Mesh must succeed.

Data Mesh must *learn from the past solutions* and address their shortcomings. It must *reduce points of centralization* that act as coordination bottlenecks. It must find a new way in *decomposing the data architecture* that, unlike technology-driven decomposition, does not slow the organization down with multi-point synchronizations. It must remove the gap between where the data originates and where it gets used in its analytical form, and *remove all the accidental complexities* - aka pipelines - that happen in between the two planes of data. Data Mesh must depart from data myths such as a single source of truth, or one tightly-controlled canonical model.

Ultimately, Data Mesh's *goal is to enable organizations to thrive* in the face of the *growth of data sources*, *growth of data users* and *use cases,* and the increasing change in cadence and complexity. Adopting Data Mesh, organizations must thrive in *agility, creating data-driven value* while *embracing change*.

Figure 2-1 lists the expected organizational outcomes applying Data Mesh, as the organization size and complexity grows, as the diversity of data and organization's data aspirations scale.

Figure 2-1. Data Mesh outcomes for organizations

In this chapter we look at the top-level outcomes that your organization achieves by adopting Data Mesh, the impact of Data Mesh and why you should care about it. For each of these outcomes, I discuss how Data Mesh accomplishes them, what shifts it creates. In discussing the shifts I give you a brief description of the foundational principles of Data Mesh -- *Domain Data Ownership, Data as a Product, Self-serve Data Platform, Computational Federated Governance*.You will see these in action and I point you to Part II and Part III of the book where you can get all the details.

Embrace Change in a Complex, Volatile and Uncertain Business Environment

Businesses are complex systems, composed of many domains that each have their own accountability structure, goals, and each changing at a different pace. The behavior of the business as a whole is the result of an intricate network of relationships between its domains and functions, their interactions and dependencies. The volatility and rapid change of the markets and regulations within which the businesses operate compounds the complexity.

How can businesses manage the impact of such complexity on their data? How can organizations keep going through change while continuing to get value from their data? How can businesses avoid increased cost of managing the change of their data landscape? How can they provide truthful and trustworthy data without disruption, in the face of continuous change? This comes down to *embracing change* in a complex organization.

In this section I discuss a few ways Data Mesh achieves embracing change despite increased complexity of the business.

Align Business, Tech and Now Analytical Data

One way to manage complexity is to break it down into independently managed parts . Businesses do this by creating domains. For example, Daff Inc. breaks down its business domains according to relatively independent outcomes and functions--including managing podcasts, managing artists, player applications, playlists, payments, marketing, etc.

This allows the domains to move fast without tight synchronization dependencies to other parts of the business.

Just as a business divides its work through business domains, technology can, and should, align itself to these business divisions. We see the best organizations orienting their technology staff around their business units, allowing each business unit to be supported by a dedicated technology capability for that unit's work. The recent movement towards Microservices is largely about performing this kind of decomposition. As part of this we see business units controlling and managing their operational applications and data.

The first principle of data mesh carries out the same decomposition for analytic data, resulting in the *Domain Ownership of Data*. Each business unit takes on the responsibility for analytic data ownership and management. This is because the people who are closest to the data are best able to understand what analytic data exists, and how it should best be interpreted.

Domain ownership distribution results in a distributed data architecture, where the data artifacts - datasets, code, metadata, and data policies - are maintained by their corresponding domains

Figure 2-2 is demonstrating the concept of organizing technology (services), analytical data aligned with business.

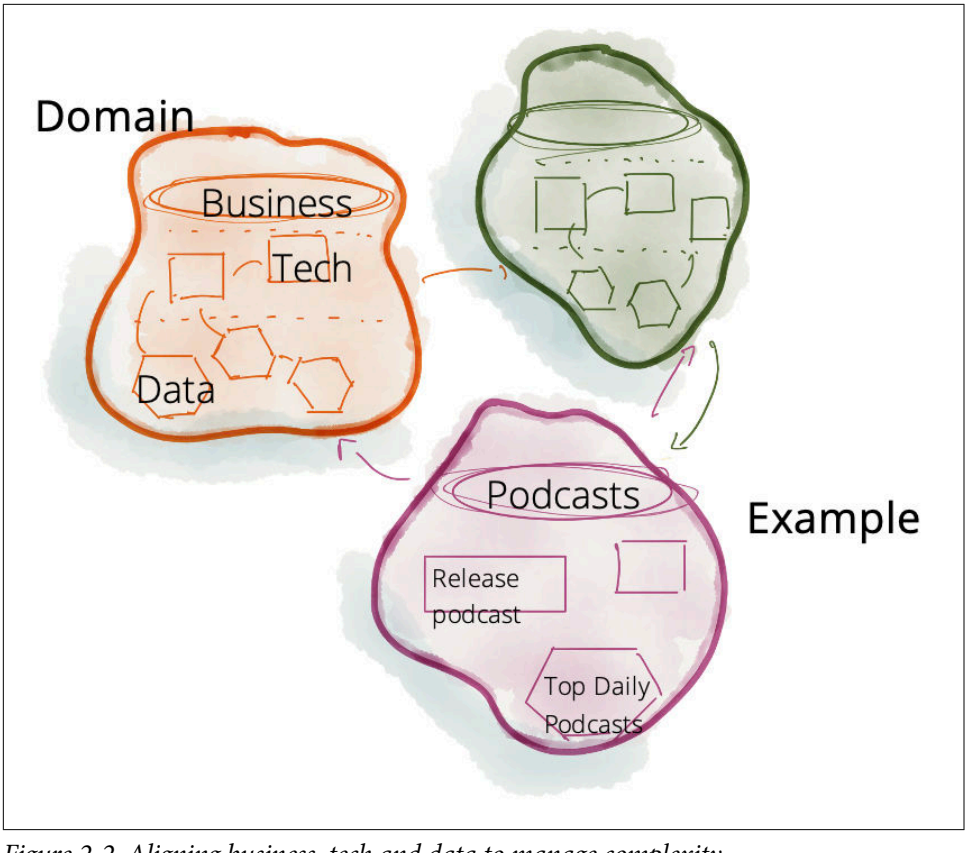

Figure 2-2. Aligning business, tech and data to manage complexity

See Chapter 4, The Principle of Domain Ownership for further details.

Close The Gap Between Analytical and Operational Data

To make good decisions in the moment, analytical data must reflect *business truthfulness*. They must be as close as possible to the facts and reality of the business at the moment the decision is made. As we saw in Chapter 1, this can't be achieved with two separate data planes - analytical and operational data planes - that are far from each other and connected through fragile data pipelines and intermediary data teams. Data

pipelines must dissolve and give way to a new way of providing the analytical data and capabilities as close to the source as possible.

Changes in the business, such as adding a new feature to a product, introducing a new service, or optimizing a workflow, must be reflected near real time in both the state of the business captured by operational data as well as its temporal view captured by the analytical data.

Data Mesh suggests that we continue to recognize and respect the differences between these two planes: the nature and topology of the data, the differing use cases, their unique personas of consumers, and ultimately their diverse access patterns. However Data mesh connects these two planes under a different structure - *an inverted model and topology based on domains and not technology stack* - where each domain extends its responsibilities to not only provide operational capabilities but also serve and share *analytical data as a product.*

Data Mesh principles of *Data as a Product* introduces an accountability for the domains to serve their analytical data as a product and delight the experience of data consumers; streamlining their experience discovering, understanding, trusting, and ultimately using quality data. Data as a product principle is designed to address the data quality and the age-old siloed data problem, and their unhappy data consumers. See Chapter 6, The Principle of Data as a Product for more on this.

Implementing this approach introduces a new architectural unit, called data product quantum that will embed all the structural components needed to maintain and serve data as a product. The structural components include the code that maintains the data, additional information, metadata, to make data discoverable and usable, and a contract to access the data in a variety of access modes native to the data consumers.

Figure 2-3 demonstrates a different integration model between operational and analytical planes. You have seen these planes in chapter 1, integrated through clever and complex data pipelines. Here, the planes are divided by business domains. The integration between data product quantums, the analytical data plane, and their corresponding domain's operational plane services are rather simple and unintelligent. A simple movement of data. Data product quantums will embed and abstract the intelligence and code required to transform the operational data into its analytical form.

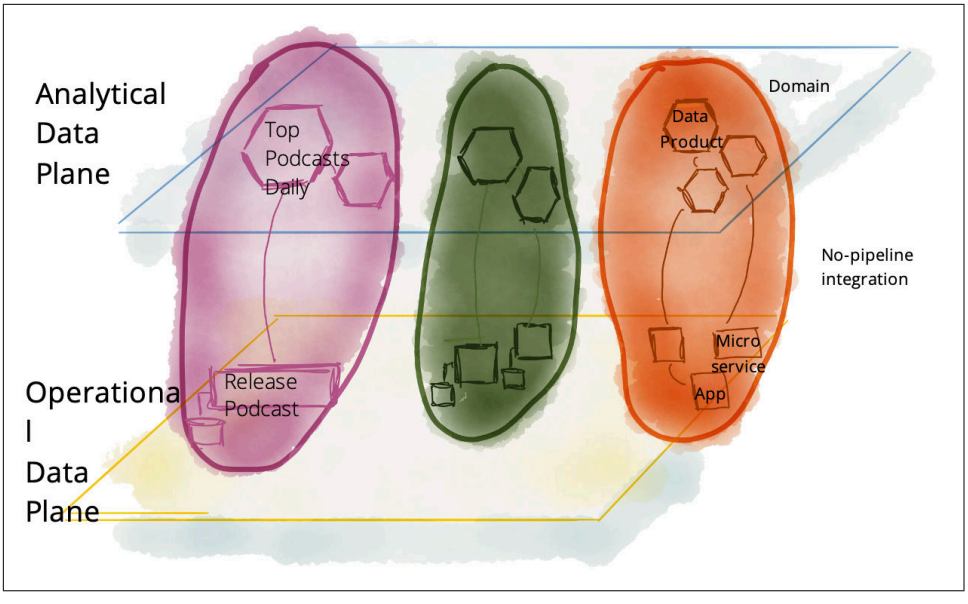

Figure 2-3. Closing the gap between operational and analytical data

Differences in today's available technology to manage the two archetypes of data should not lead to the separation of organizations, teams, and people who work on them. I believe that our technologies will evolve at some point in the future to bring these two planes even closer together, but for now, I suggest we keep their concerns separate. The primary focus of this book, and Data Mesh itself, is on the analytical plane and its integration with the operational plane.

Localize Data Change to Business Domains

Data Mesh must allow for data models to change continuously without fatal impact to downstream data consumers, or slowing down access to data as a result of synchronizing change of a shared global canonical model. Data Mesh achieves this by localizing change to domains by providing autonomy to domains to model their data based on their most intimate understanding of the business without the need for central coordinations of change to a single shared canonical model.

Data Mesh imposes contracts, well-defined and guaranteed interfaces, to share data. This liberates domains to change their data models, given that they still support the older revisions of their contracts, until they gracefully migrate their data users to the new revisions. Data Mesh introduces a set of discovery APIs that allow data product users to locate and consume data according to the guarantees of the data discovery APIs. See Chapter 4, The Principle of Domain Ownership and Chapter 5, The Principle of Data as a Product for more on this.

Reduce Accidental Complexity of Pipelines and Copying Data

As Fred Brooks laid out in his widely popular paper, "No Silver Bullet – Essence and Accident in Software Engineering", there are two types of complexity when building software systems. First, the essential complexity: the complexity that is essential and inherent to the problem space. This is the business and domain complexity we discussed earlier. And second, the accidental complexity: the complexity that we - engineers, architects and designers - create in our solutions and can be fixed.

The world of analytical solutions is full of opportunities to remove and fix accidental complexities. Let's talk about a few of those accidental complexities that Data Mesh must reduce.

Today, we keep copying data around because we need the data for yet another mode of access, or yet another model of computation. We copy data from operational systems to a data lake for data scientists. We copy the data again into lakeshore marts for data analyst access and then into the downstream dashboard or reporting databases for the last mile. We build complex and brittle pipelines to do the copying. The copying journey continues across one technology stack to another and across one cloud vendor to another. Today, to run analytical workloads you need to decide upfront which cloud provider copies all of your data in its lake or warehouse before you can get value from it.

Data Mesh addresses this problem by creating a new architectural unit that encapsulates a domain-oriented data semantic, but yet provides multiple modes of access to the data suitable for different use cases and users. This architectural unit is called the Data Product Quantum. It will have a clear contract and guarantees to its readers, and meet their native access mode, SQL, files, events, etc. Data product quantum can be accessed anywhere across the internet and it provides access control and policy enforcement necessary at the time of access, locally at its interface. Data product quantum encapsulates the code that transforms and maintains its data. With abstraction of transformation code inside a data product quantum, and accessing data through data product quantum interfaces, the need for pipelines will go away. Removing the brittle concept of pipeline reduces the opportunity for failure in case of an upstream data change. Data Mesh introduces standardized interfaces to discover and access every data product enabled by a self-serve infrastructure. See Chapter 8 on the logical architecture of Data Mesh, and Chapter 9 for more details on the data product quantum and Chapter 6, on self-serve data infrastructure.

Sustain Agility in the Face of Growth

Today, businesses' successes are predicated on multi-faceted growth--new acquisitions, new service lines, new products, geolocation expansions and so on. All this leads to new sources of data to manage and new data-driven use cases to build. Many

organizations slow down or plateau in the speed of delivering value from their data, onboarding new data, or serving the use cases as they grow.

Data Mesh's approach to sustain agility in the face of growth can be summarized in a few techniques that aim to reduce bottlenecks, coordination, and synchronization. Agility relies on business domains' ability to achieve outcomes autonomously and with minimal dependencies.

Remove Centralized and Monolithic Bottlenecks of the Lake or the Warehouse

A centralized data team, managing a monolithic data lake or warehouse limits agility, particularly as the number of sources to onboard or number of use cases grow. Data Mesh looks carefully for centralized bottlenecks, particularly where they are the focal point of multi-party synchronization, both from the human communication perspective and architecture. These bottlenecks include data lakes and data warehouses.

Data Mesh proposes an alternative, a peer-to-peer approach in data collaboration when serving and consuming data. The architecture enables consumers to directly discover and use the data right from the source. For example, an ML training function or a report, can directly access independent data products, without the intervention of a centralized architectural component such as a lake or a warehouse, and without the need for an intermediary data (pipeline) team. See chapter 9 for details of peer-to-peer data consumption through data product quantum's output data ports and input data ports.

Figure 2-4 demonstrates the conceptual shift. Each data product provides versioned interfaces that allow peer-to-peer consumption of domain-data to compose aggregates or higher-order data products.

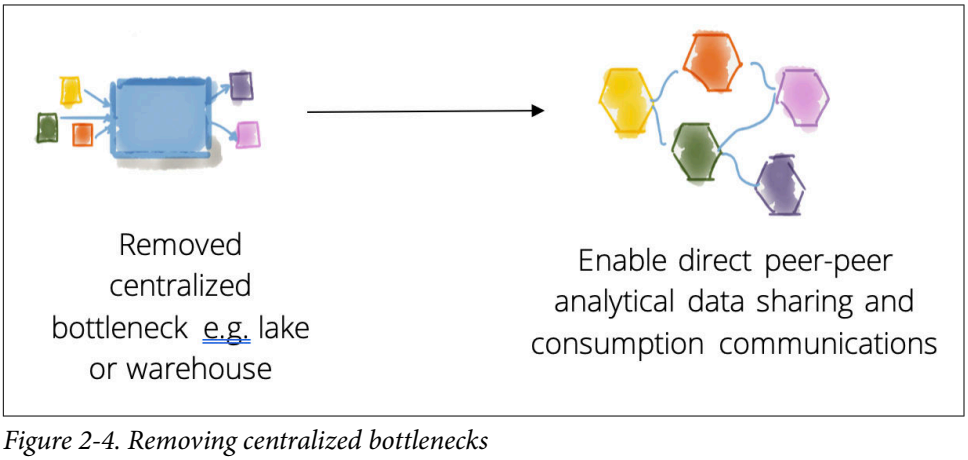

Figure 2-4. Removing centralized bottlenecks

Reduce Coordination of Data Pipelines

Over the last decades, the technologies that have exceeded in their operational scale have one thing in common, they have minimized the need for coordination and synchronization. Asynchronous IO has scaled the throughput of networked applications over blocking IO. Reactive applications, for example, have resulted in faster parallel processing of messages. Apache Hadoop scaled data processing by running Map-Reduce functional programming model across many servers distributedly. Using choreographed event-driven microservices over centrally orchestrated ones has allowed us to scale our business workflows.

Despite our relentless effort to remove coordination and synchronization from our core technologies in order to achieve scale and speed, we have, for the most part, neglected organizational and architectural coordination. As a result, no matter how fast our computer systems run, achieving outcomes have fallen behind coordinating activities of teams and humans.

Data Mesh addresses two of the main coordination issues in the data management processes and architecture. A technically-divided architecture of a pipeline - ingestion, processing, serving, etc. - results in the coordination of these functions to deliver a new data source, or serve a new use case. Data Mesh moves away from technical-partitioning of data management, to domain-oriented partitioning. Domain-oriented data products must be able to develop and evolve independently of other data products. The domain-oriented decomposition reduces the need for coordination to achieve an outcome. For the most part, a new data source or a new use case can be served by a domain-oriented data product team. In cases where a new use case requires access to a new data product outside of the domain, the consumer can make progress by utilizing the standard contracts of the new data product, mocks, stubs, or synthetic data interfaces until the data product becomes available. This is the beauty of contracts, easing the coordination between consumer and provider during development. See chapter 4, The Principle of Domain Ownership for more on this.

Reduce Coordination of Data Governance

Today, another major coordination bottleneck is the central function of data governance. Data governance coordination is necessary to provide access to data users, approve the quality of datasets, and validate the conformance of data changes with the organization's policies.

Data Mesh introduces a *federated and computational data governance model*, where the governance team is composed of the individual domain data product owners, the main owners of the data products. The governance function aims to embed policy execution into every data product in a computational and automated fashion. This vastly improves the function of governance today, which is one of the main synchro-

nization points for discovering data, approving data, and making sure it follows the necessary policies.

As you can imagine, the autonomy of the domains can have undesirable consequences if not checked; isolation of domains, incompatibility and disconnection of one domain's data product from others, and a fragmented experience when consuming multiple domains' data. Data Mesh governance heavily relies on the automation of governance concerns for a consistent, connected and trustworthy experience using the domains' data products.

See Chapter 7, The Principle of Federated Computational Governance.

Enable Autonomy

The correlation between team autonomy and team performance has been the subject of team management studies. Empirical studies show that teams' freedom in decision making to fulfill their mission can lead to better team performance. On the other hand, too much autonomy can result in inconsistencies, duplicated efforts and team isolation.

Data Mesh attempts to strike a balance between team autonomy and inter-term interoperability and collaboration, with a few complementary techniques. It gives domain teams autonomy to have control of their local decision making, such as choosing the best data model for their data products. While it uses the computational governance policies to impose a consistent experience across all data products; for example, standardizing on the data modeling language that all domains utilize.

Domain teams are given autonomy to build and maintain the lifecycle of their data products. While Data Mesh places a domain-agnostic data platform team who empowers the domain teams with self-serve capabilities to manage the lifecycle of data products declaratively and consistently, to prevent team isolation and decrease cost of autonomy.

The *principles of self-serve data platform*, essentially makes it feasible for domain teams to manage the lifecycle of their data products with *autonomy*, and utilize the skillsets of their generalist developer to doso.

These self-serve data infrastructure APIs allow data product developers to build, deploy, monitor and maintain their data products. The APIs allow data consumers to discover, learn, access, and use the data products. The self-serve infrastructure makes it possible for the mesh of data products to be joined, correlated and used as a whole, while maintained independently by the domain teams.

See Chapter 6, The Principle of Self-serve Data Platform, for more.

Increase the Ratio of Value from Data to Investment

Industry reports, such as the NewVantage Partner I shared in chapter 1, and my personal experience, point to the fact that we are getting little value from data compared to the investments we are making in data management. If we compare the value we get from our data teams and data solutions, compared to other technical developments such as infrastructure, mobile and application development, it's evident that we are facing headwinds when it comes to data.

Data Mesh looks at ways to improve the ratio of value over effort in analytical data management: creation of a new archetype of data platform that abstracts today's technical complexity, through open data interfaces that enable sharing data across organizational trust boundary or physical location, and through applying product thinking to remove friction across all stages of the value stream that gets analytical data from the points of origin to the hands of data scientists and analysts.

Abstract Technical Complexity with a Data Platform

Today's landscape of data management technology is undoubtedly too complex. The litmus test for technical complexity is the ever growing need for *data engineers* and *data technology experts*. We don't seem to ever have enough of them. Another litmus test is the low value to effort ratio of data pipeline projects. Much effort is spent with little value returned - getting access to baseline datasets with quality.

Data Mesh looks critically at the existing technology landscape, and reimagines the technology solutions as a *data-product-developer*(or consumer)-*centric* platform. In chapter 6, The Principle of Self-Serve Data Platform, we will see how Data Mesh arrives at a set of self-serve data platform capabilities to remove friction and complexity from the workflows of *data product developers* and *data product users*. Data Mesh intends to remove the need for data specialists and enable generalist experts to develop data products.

Additionally, Data Mesh defines a set of open interfaces for different affordances of data products - discovering, querying, serving data, etc. - to enable a more collaborative ecosystem of tools. This is in contrast to a heavily proprietary data technology landscape with a high cost of integration across vendors. See chapter 9 on data products' open interfaces.

Embed Product Thinking Everywhere

Data Mesh introduces a few shifts to get us laser focused on the value, as perceived by the data users. It shifts our thinking from data as an *asset* to data as a *product*. It shifts how we measure success from the volume of the data to measure to the happiness and satisfaction of the data users. See chapter 5, The Principle of Data as a Product, for more details on achieving this shift.

Data is not the only component of a Data Mesh ecosystem that is treated as a product. The self-serve data platform itself is also a product. In this case, it serves the data product developers and data product consumers. Data Mesh shifts the measure of success of the platform from the number of its capabilities, to the impact of its capabilities on improving the experience of data product development, the reduced lead time to deliver, or discover and use of a data product. See chapter 5, The Principle of Self-Serve Data Platform for more on this.

Go Beyond The Boundaries

The value that a business unit can generate almost always requires data beyond the unit's boundary, requiring data that comes from many different business domains. Similarly, the value that an organization can generate serving its customers, employees, and partners often requires access to data beyond the data that the organization generates and controls.

Consider Daff Inc. In order to provide a better experience to the listeners with autoplay music, it not only requires data from listeners' playlist, but also their network, as well as their social and environmental influences and behaviors. It requires data from many corners of Daff Inc, and beyond including news, weather, social platforms, etc.

Multi-domain and multi-org access to data is an assumption built into Data Mesh. Data Mesh's data product concept can provide access to data no matter where the data physically resides. Data product quantum provides a set of interfaces that essentially allow anyone with the proper access control, discover, and use the data product independent of its physical location. The identification schema, access control and other policy enforcement assumes using open protocols that can be enabled over the internet. See chapter 8, Data Mesh Logical Architecture for more on this.

Recap

After reading this chapter you might assume that Data Mesh is a silver bullet. Quite the contrary. Data Mesh is an important piece of the puzzle. It enables us to truly democratize access to data. However to close the loop of deriving value from data, there is much more that needs to be done beyond just getting access to data. We must be able to continuously deliver analytical and ML-based solutions. However bootstrapping this closed loop, requires access to data at scale, which is a focus of Data Mesh.

The Data Mesh goals listed in this chapter, invites us to *reimagine* data, how to architect solutions to manage it, how to govern it, and how to structure our teams . The expectation to *become resilient to business complexity, to sustain agility in the face of growth,* and *accelerate getting value from data* pushes us to work with the reality of increasing complexity, growth and volatility instead of fighting to control it.

In the next chapter, I will give an overview of what has happened before the inflection point. Why the data management approach that got us here, won't take us to the future.

CHAPTER 3
Before The Inflection Point

A note for Early Release readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

> *Today's problems come from yesterday's "solutions."*
>
> —Senge, Peter M. The Fifth Discipline

Organizational complexity, growth of data sources, proliferation of data expectations. These are the forces that have put stress on our existing approaches to analytical data management. Our existing methods have made remarkable progress scaling the machines: manage large volumes of a variety of data types with planet scale distributed data storage, reliably transmit high velocity data through streams, and process data intensive workloads, concurrently and fast. However, our methods have limitations with regard to the organizational complexity and scale, the human scale.

In this chapter, I give a short introduction to the current landscape of data architectures, their underlying characteristics and the reasons why, moving into the future, they limit us.

Evolution of Analytical Data Architectures

How we manage analytical data has gone through evolutionary changes; changes driven by new consumption models, ranging from traditional analytics in support of business decisions to intelligent business functions augmented with ML. While we have seen an accelerated growth in the number of analytical data technologies, the *high level architecture* has seen very few changes. Let's have a quick browse of the high level analytical data architectures, followed by a review of their unchanged characteristics.

The underlying technologies supporting each of the following architectural paradigms have gone through many iterations and improvements. The focus here is on the architectural pattern, and not the technology and implementation evolutions.

First Generation: Data Warehouse Architecture

Data warehousing architecture today is influenced by early concepts such as facts and dimensions formulated in the 1960s. The architecture intends to flow data from operational systems to business intelligence systems that traditionally have served the management with operations and planning of an organization. While data warehousing solutions have greatly evolved, many of the original characteristics and assumptions of their architectural model remain the same:

- Data is extracted from many operational databases and sources
- Data is transformed into a universal schema - represented as a multi-dimensional and time-variant tabular format
- Data is loaded into the warehouse tables
- Data is accessed through SQL-like querying operations
- Data is mainly serving data analysts for their reporting and analytical visualizations use cases

The data warehouse approach is also referred to as data marts with the usual distinction that a data mart serves a single department in an organization, while a data warehouse serves the larger organization integrating across multiple departments. Regardless of their scope, from the architectural modeling perspective they both have similar characteristics.

In my experience, the majority of enterprise data warehouse solutions are proprietary, expensive and require specialization for use. Over time, they grow to thousands of ETL jobs, tables and reports that only a specialized group can understand and maintain. They don't let themselves to modern engineering practices such as CI/CD and

incur technical debt over time and an increased cost of maintenance. Organizations attempting to escape this debt, find themselves in an inescapable cycle of migrating from data warehouse solution to another.

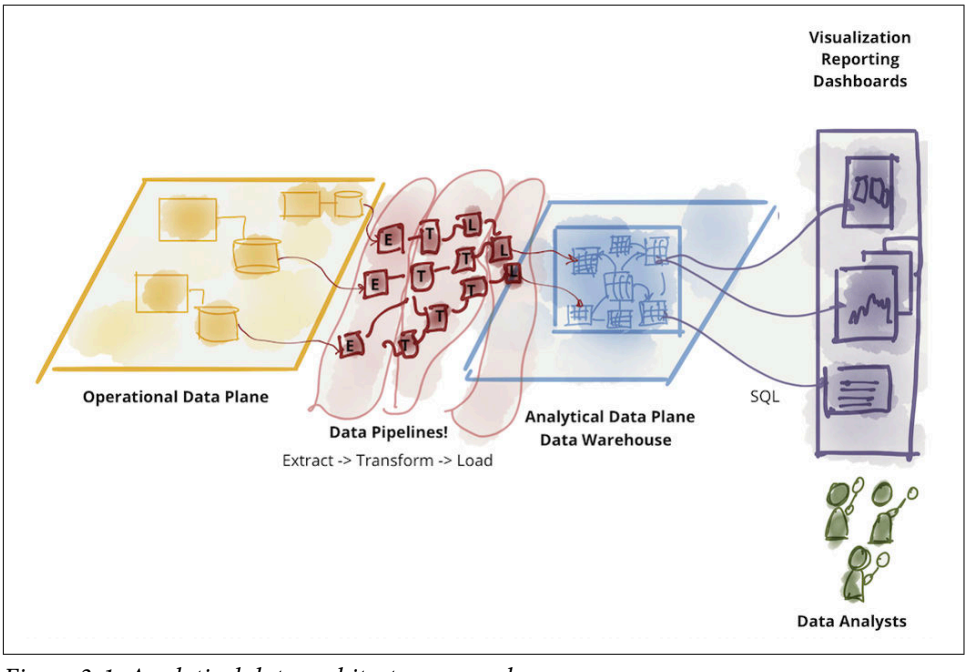

Figure 3-1. Analytical data architecture - warehouse

Second Generation: Data Lake Architecture

Data lake architecture was introduced in 2010 in response to challenges of data warehousing architecture in satisfying the new uses of data; access to data based on data science and machine learning model training workflows, and supporting massively parallelized access to data. Data lake architecture, similarly to data warehouse, assumes that data gets extracted from the operational systems and is loaded into a central repository often in the format of an object store, storage of any type of data. However unlike data warehousing, data lake assumes no or very little transformation and modeling of the data upfront; it attempts to retain the data close to its original form. Once the data becomes available in the lake, the architecture gets extended with elaborate transformation pipelines to model the higher value data and store it in lakeshore marts.

This evolution to data architecture aims to improve ineffectiveness and friction introduced by extensive upfront modeling that data warehousing demands. The upfront transformation is a blocker and leads to slower iterations of model training. Additionally, it alters the nature of the operational system's data and mutates the data in a

way that models trained with transformed data fail to perform against the real production queries.

In our example, a music recommender when trained against a transformed and modeled data in a warehouse, fails to perform when evoked in an operational context - e.g. evoked by the recommender service with the logged-in user's session information. The heavily transformed data used to train the model, either misses some of the user's signals or has created a different representation of users attributes. Data lake comes to rescue in this scenario.

Notable characteristics of a data lake architecture include:

- Data is extracted from many operational databases and sources
- Data is minimally transformed to fit the storage format e.g. Parquet, Avro, etc.
- Data - as close as the source syntax - is loaded to scalable object storage
- Data is accessed through the object storage interface - read as files or data frames - a two-dimensional array-like structure.
- Lake storage is accessed mainly for analytical and machine learning model training use cases and used by data scientists
- Downstream from the lake, lake shore marts, are fit-for-purpose data marts or data services serve the modeled data
- Lakeshore marts are used by applications and analytics use cases

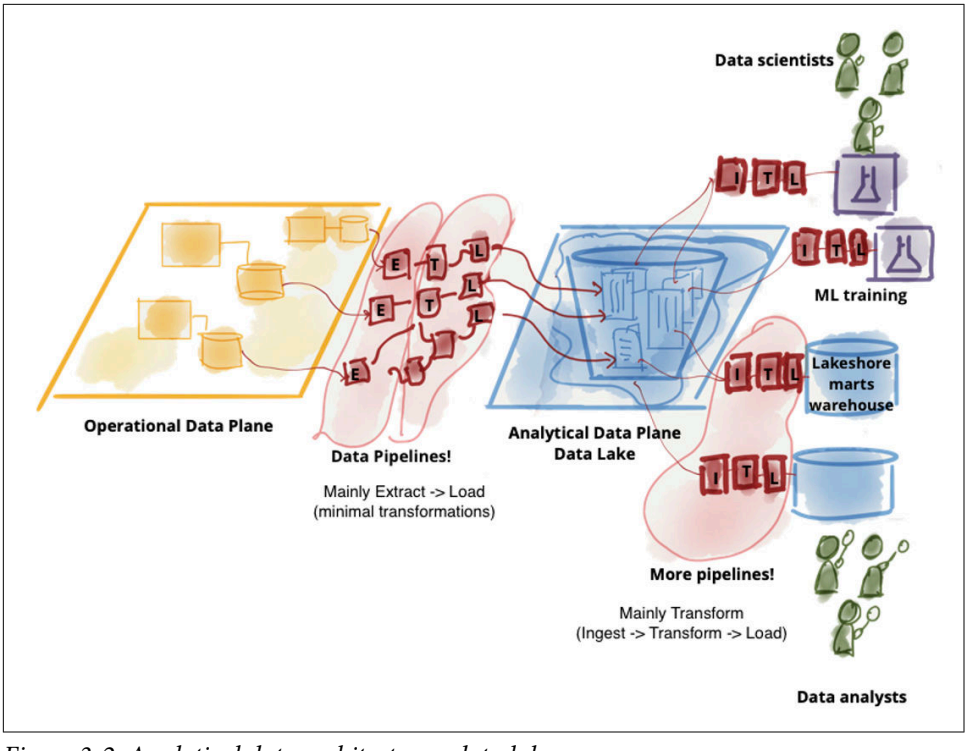

Figure 3-2. Analytical data architecture - data lake

Data lake architecture suffers from complexity and deterioration; complex and unwieldy pipelines of batch or streaming jobs operated by a central team of hyper-specialized data engineers; deteriorated and unmanaged datasets, untrusted, inaccessible, provide little value.

Third Generation: Multimodal Cloud Architecture

The *third and current generation data architectures* are more or less similar to the previous generations, with a few modern twists:

- Streaming for *real-time data availability* with architectures such as Kappa
- Attempting to unify the batch and stream processing for data transformation with frameworks such as Apache Beam
- Fully embracing cloud based managed services with modern cloud-native implementations with isolated compute and storage
- Convergence of warehouse and lake, either extending data warehouse to include embedded ML training, e.g. Google BigQuery ML, or alternatively build data

warehouse integrity, transactionality and querying systems into data lake solutions, e.g., Databricks Lakehouse

The third generation data platform is addressing some of the gaps of the previous generations such as *real-time data analytics*, as well as *reducing the cost of managing big data infrastructure*. However it suffers from many of the underlying characteristics that have led to the limitations of the previous generations.

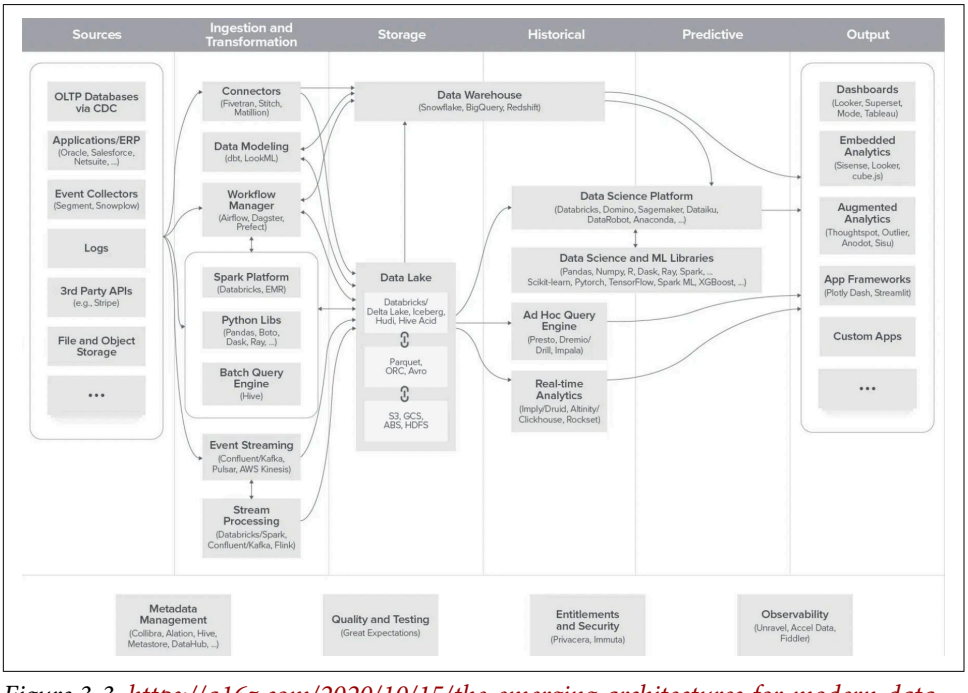

Figure 3-3. https://a16z.com/2020/10/15/the-emerging-architectures-for-modern-data-infrastructure Multimodal data architecture

Characteristics of Analytical Data Architecture

From the quick glance at the history of analytical data management architecture, it is apparent that the architecture has gone through evolutionary improvements. The technology and products landscape in support of the data management have gone through a cambrian explosion and continuous growth. The dizzying view of First-Mark's[1] annual landscape and "state of the union" in big data and AI, is an indication of the sheer number of innovative solutions developed in this space.

1 An early-stage venture capital firm in New York City

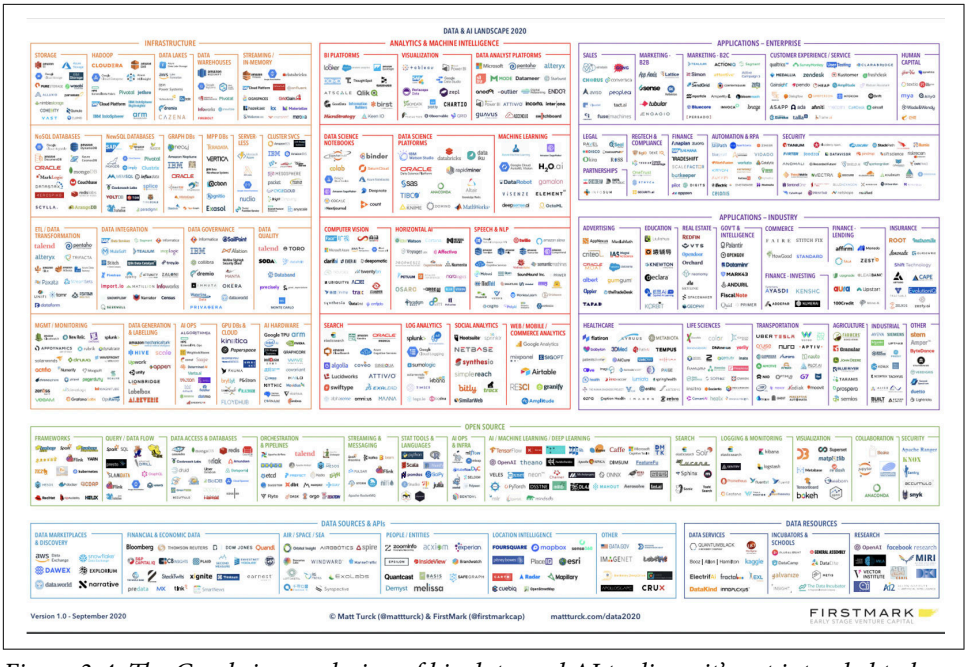

Figure 3-4. The Cambrian explosion of big data and AI tooling - it's not intended to be read, just glanced and feel dizzy Courtesy of FirstMark

So the question is, what hasn't changed? What are the underlying characteristics that all generations of analytical data architecture carry? Despite the undeniable innovation, there are fundamental assumptions that have remained unchallenged for the last few decades and must be closely evaluated:

- Data must be centralized to be useful - managed by a centralized organization, with an intention to have an enterprise-wide taxonomy.
- Data management architecture, technology and organization are monolithic.
- The enabling technologies dictate the paradigm - architecture and organization.

The architectural characteristics discussed in this chapter, including centralization, are only applied to the logical architecture. Physical architecture concerns such as where the data is physically stored - whether it is physically collocated or not - is out of scope for our conversation, and it's independent of the logical architecture concerns. The logical architecture focuses on the experience layer of the data developers and consumers. Such as whether data is being managed by a single team or not - data ownership - whether data has a single schema or not - data modeling - and whether a change on one data model has tight coupling and impact on downstream users - dependencies.

Let's look a bit more closely at each of these underlying assumptions and the limitations each impose.

Monolithic

> Architecture styles can be classified into two main types: monolithic (single deployment unit of all code) and distributed (multiple deployment units connected through remote access protocols)
>
> —Fundamentals of Software Architecture

Monolithic Architecture

At 30,000 feet the data platform architecture looks like Figure 2-7 below; a monolithic architecture whose goal is to:

- Ingest data from all corners of the enterprise and beyond, ranging from operational and transactional systems and domains that run the business, to external data providers that augment the knowledge of the enterprise. For example in the case of Daff Inc., the data platform is responsible for ingesting a large variety of data: the 'media players performance', how their 'users interact with the players', 'songs they play', 'artists they follow', 'labels' and 'artists' that the business has onboarded, the 'financial transactions' with the artists, and external market research data such as 'customer demographic' information.
- Cleanse, enrich, and transform the source data into trustworthy data that can address the needs of a diverse set of consumers. In our example, one of the transformations turns the 'user clicks stream" to 'meaningful user journeys' enriched with details of the user. This attempts to reconstruct the journey and behavior of the user into an aggregate longitudinal view.
- Serve the datasets to a variety of consumers with a diverse set of needs. This ranges from data exploration, machine learning training, to business intelligence reports. In the case of Daff Inc., the platform must serve 'media player's near real-

time errors' through a distributed log interface and at the same time serve the batched aggregate view of a particular 'artist played record' to calculate the monthly financial payments.

Figure 3-5. The 30,000 ft view of the monolithic data platform

While a monolithic architecture can be a good and a simpler starting point for building a solution - e.g. managing one code base, one team - it falls short as the solution scales. The drivers we discussed in Chapter 1, organizational complexity, proliferation of sources and use cases, create tension and friction on the architecture and organizational structure:

- Ubiquitous data and source proliferation: As more data becomes ubiquitously available, the ability to consume it all and harmonize it in one place, logically, under the control of a centralized platform and team diminishes. Imagine the domain of 'customer information'. There are an increasing number of sources inside and outside of the boundaries of the organization that provide information about the existing and potential customers. The assumption that we need to ingest and harmonize the data under a central customer master data management to get value, creates a bottleneck and slows down our ability to take advantage of diverse data sources. The organization's response to making data available from new sources slows down as the number of sources increases.
-
- Organizations' innovation agenda and consumer proliferation: Organizations' need for rapid experimentation introduces a larger number of use cases that consume the data from the platform. This implies an ever growing number of transformations to create data - aggregates, projections and slices that can satisfy the test and learn cycle of innovation. The long response time to satisfy the data consumer needs has historically been a point of organizational friction and remains to be so in the modern data platform architecture. The disconnect between peo-

ple and systems who are in need of the data and understand the use case, from the actual sources, teams and systems, who originated the data and are most knowledgeable about the data, impedes the company's data-driven innovations. It lengthens the time needed to access the right data, and becomes a blocker for hypothesis-driven development.

-
- Organizational complexity: Adding a volatile and continuously shifting and changing data landscape - data sources and consumers - to the mix, is when a monolithic approach to data management becomes a synchronization and prioritization hell. Aligning the priorities and activities of the continuously changing data sources and consumers, with the capabilities and priorities of the monolithic solution - isolated from the sources and consumers - is a no-win situation.

Monolithic Technology

From the technology perspective, the monolithic architecture has been in a harmonious accordance with its enabling technology; technologies supporting data lake or data warehouse architecture, by default, assume a monolithic architecture. For example, data warehousing technologies such as Snowflake, Google BigQuery, or Synapse, all have a monolithic logical architecture - architecture from the perspective of the developers and users. While at the physical and implementation layer there has been immense progress in resource isolation and decomposition - for example Snowflake separates compute resource scaling from storage resources and BigQuery uses the latest generation distributed file system - the user experience of the technology remains monolithic.

Data Lake technologies such as object storage and pipeline orchestration tools, can be deployed in a distributed fashion. However by default, they do lead to creation of monolithic lake architectures. For example, data processing pipeline DAG definition and orchestrations' lack of constructs such as interfaces and contracts abstracting pipeline jobs dependencies and complexity, leads to a big ball of mod monolithic architecture with tightly coupled labyrinthic pipelines, where it is difficult to isolate change or failure to one step in the process. Some cloud providers have limitations on the number of lake storage accounts, having assumed that there will only be a small number of monolithic lake setups.

Monolithic Organization

From the organizational perspective, Conway's Law has been at work and in full swing with monolithic organizational structures - business intelligence team, data analytics group, or data platform team - responsible for the monolithic platform, its data and infrastructure.

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.
>
> —Melvin Conway, 1968

When we zoom close enough to observe the life of the people who build and operate a data platform, what we find is a group of hyper-specialized data engineers siloed from the operational units of the organization; where the data originates or where it is used. The data engineers are not only siloed organizationally but also separated and grouped into a team based on their technical expertise of data tooling, often absent of business and domain knowledge.

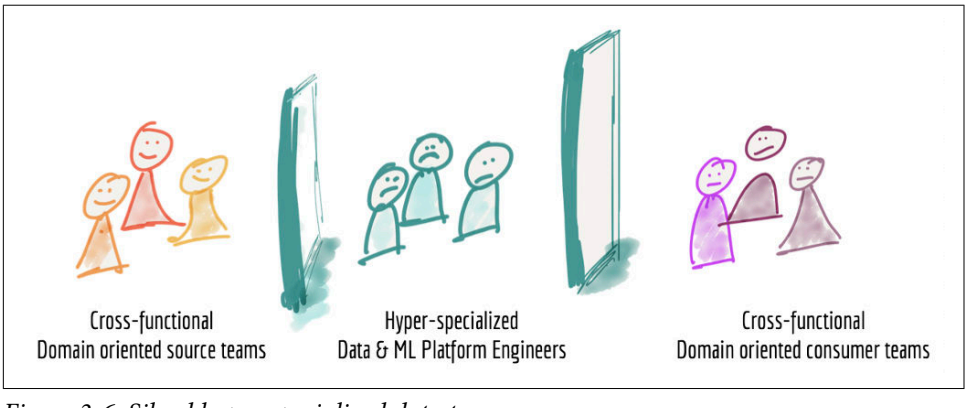

Figure 3-6. Siloed hyper-specialized data team

I personally don't envy the life of a data engineer. They need to consume data from operational teams who have no incentive in providing meaningful, truthful and correct data, based on an agreed-upon contract. Given the data team's organizational silo, data engineers have very little understanding of the source domains that generate the data and lack the domain expertise in their teams. They need to provide data for a diverse set of needs, operational or analytical, without a clear understanding of the application of the data and access to the consuming domain's experts.

For example at Daff Inc., on the source side we have a cross-functional 'media player' team that provide signals of how users interact with media player features e.g. 'play song events', 'purchase events', and 'play audio quality'; and on the other end sit a cross-functional consumer team such as 'song recommendation' team, 'sales team' reporting sales KPIs, 'artists payment team' who calculate and pay artists based on play events, and so on. Sadly, in the middle sits the data team that through sheer effort provides analytical data on behalf of all sources and to all consumers.

In reality what we find are disconnected source teams, frustrated consumers fighting for a spot on top of the data team's backlog and an over stretched data team.

The complicated monolith

Monolithic architectures when they meet scale - here, scale in diversity of sources, consumers, and transformations - all face a similar destiny, becoming a complex and difficult to manage system.

The complexity debt of the sprawling data pipelines, duct-taped scripts implementing the ingestion and transformation logics, the large number of datasets - tables or files - with no clear architectural and organizational modularity, and thousands of reports built on top of those datasets, keeps the team busy paying the interest of the debt instead of creating value.

In short, a monolithic architecture, technology and organizational structure is not suitable for analytical data management of large scale and complex organizations.

Centralized

It's an accepted convention that the monolithic data platform hosts and owns the data that belongs to different domains, e.g. 'play events', 'sales KPIs', 'artists', 'albums', 'labels', 'audio', 'podcasts', 'music events', etc.; collected from a large number of disparate domains.

While over the last decade we have successfully applied domain driven design and bounded context to the design of our operational systems to manage complexity at scale, we have largely disregarded the domain driven design paradigm in a data platform. DDD's strategic design introduces a set of principles to manage modeling at scale, in a large and complex organization. It encourages moving away from a single canonical model to many bounded contexts' models. It defines separate models each owned and managed by a unit of organization. It explicitly articulates the relationships between the models.

While operational systems have applied DDD's strategic design techniques toward *domain-oriented data ownership*, aligning the services and their data with existing business domains, analytical data systems have maintained a *centralized data ownership outside of the domains*.

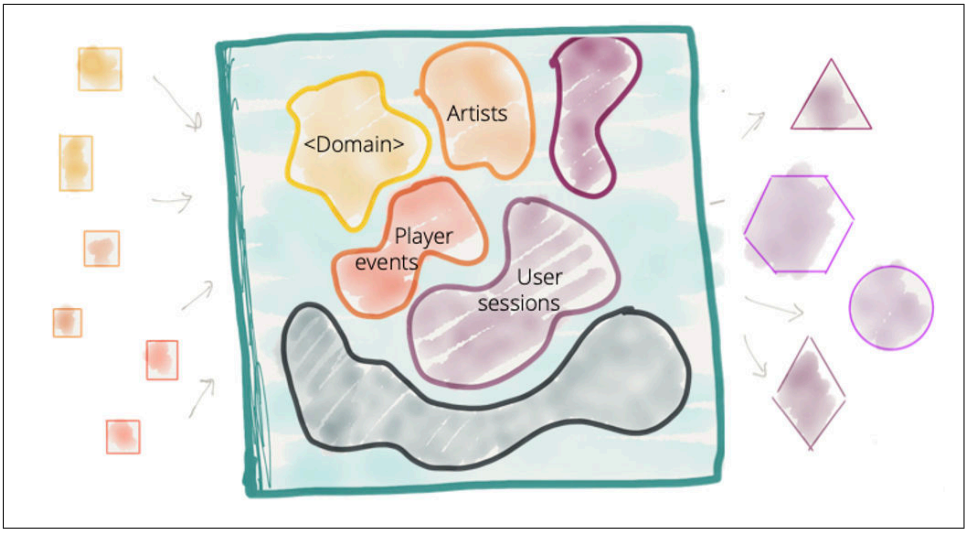

Figure 3-7. Centralization of data with no clear data domain boundaries and domain-oriented ownership of data

While this centralized model can work for organizations that have a simpler domain with a smaller number of consumption cases, it fails for enterprises with rich and complex domains.

In addition to limitations to scale, other challenges of data centralization include providing quality data that is resilient to change; data that is as closely as possible is reflective of the facts of the business with integrity. The reason for this is that business domains and teams who are most familiar with the data, who are best positioned to provide quality data right at the source, are not responsible for data quality. The central data team, far from the source of the data and isolated from the domains of the data, is tasked with building quality back into the data through data cleansing and enriching pipelines. Often, the data that pops out of the other end of the pipelines into the central system loses its original form and meaning.

Centralization of the analytical data has been our industry's response to the siloed and fragmented data, commonly known as Dark Data. Coined by Gartner, Dark Data refers to the information assets organizations collect, process and store during regular business activities, but generally fail to use for analytical or other purposes.

Technology driven

Looking back at different generations of analytical data management architectures, from warehouse to lake and all on the cloud, we have heavily leaned on a technology-driven architecture. A typical solution architecture of a data management system merely wires various technologies, each performing a technical function, a piece of an

end to end flow. This is evident from a glance at any cloud provider's modern solution architecture diagram, like the one below. The core technologies listed below are powerful and helpful in enabling a data platform. However, the proposed solution architecture decomposes and then integrates the components of the architecture based on their technical function and the technology supporting the function. For example, first we encounter the *ingestion* function supported by Cloud Pub/Sub, then *publishing data* to Cloud Storage which then *serves data* through BigQuery. This approach leads to a *technically-partitioned architecture* and consequently *an activity-oriented team decomposition.*

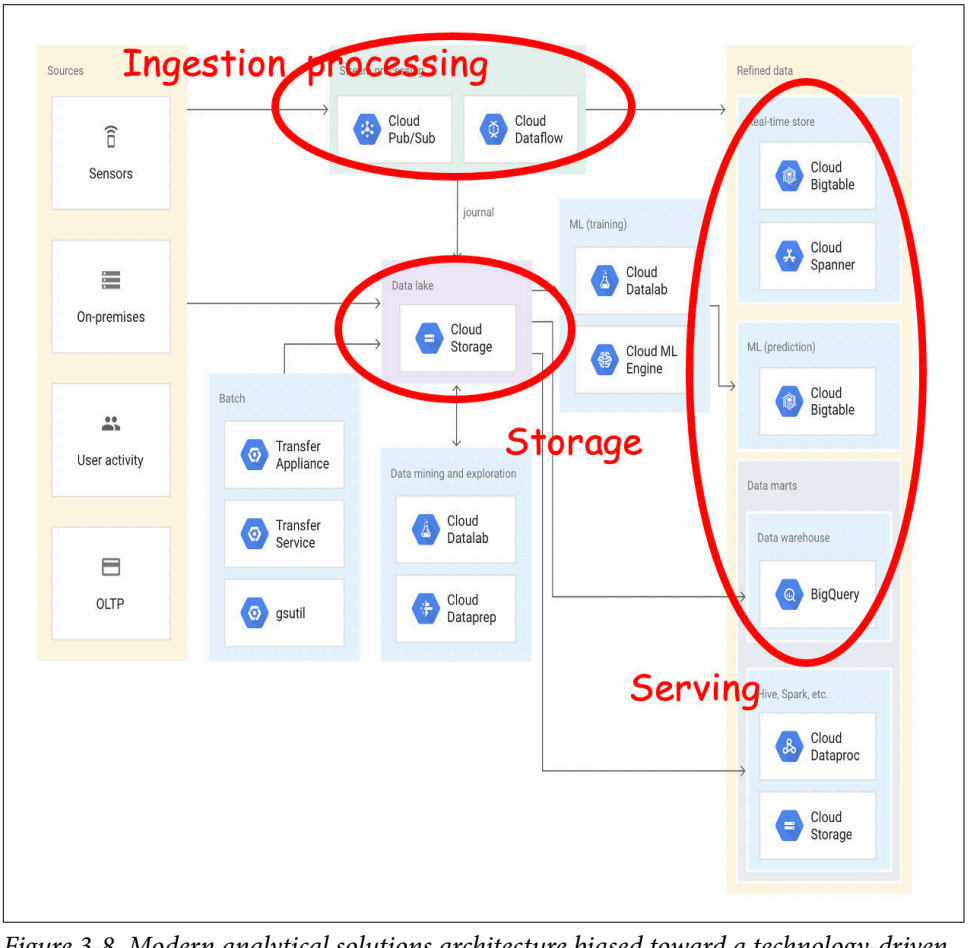

Figure 3-8. Modern analytical solutions architecture biased toward a technology-driven decomposition - example from GCP https://cloud.google.com/solutions/build-a-data-lake-on-gcp

Technically-Partitioned Architecture

One of the limitations of data management solutions today, comes down to how we have attempted to manage its unwieldy complexity; how we have decomposed an ever-growing monolithic data platform and team to smaller partitions. We have chosen the path of least resistance, a *technical partitioning for the high level architecture.*

Architects and technical leaders in organizations decompose an architecture in response to its growth. The need for on-boarding numerous new sources, or responding to proliferation of new consumers requires the platform to grow. Architects need to find a way to scale the system by breaking it into its top-level components.

Top-level technical partitioning, as defined by Fundamentals of Software Architecture, decomposes the system into its components based on their technical capabilities and concerns; it's a decomposition that is closer to the implementation concerns than business domain concerns. Architects and leaders of monolithic data platforms have decomposed the monolithic solutions based on a pipeline architecture, into its technical functions such as *ingestion*, *cleansing*, *aggregation*, *enrichment, and serving.* The top-level functional decomposition leads to synchronization overhead and slow response to data changes, updating and creating new sources or use cases. An alternative approach is a *top-level domain-oriented top-level partitioning,* where these technical functions are embedded to the domain, where the change to the domain can be managed locally without top-level synchronization.

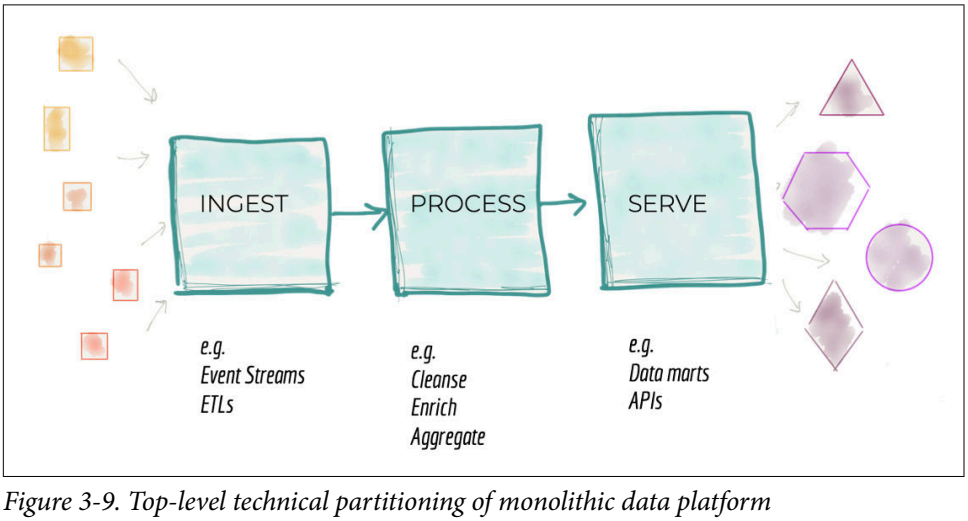

Figure 3-9. Top-level technical partitioning of monolithic data platform

Activity-oriented Team Decomposition

The motivation behind breaking a system down into its architectural components is to create independent teams who can each build and operate an architectural component. These teams in turn can parallelize work to reach higher operational scalability and velocity. The consequence of top-level technical decomposition is decomposing teams into *activity-oriented groups*, each focused on a particular activity required by a stage of the pipeline. For example, a team focusing on *ingestion* of data from various sources or a team responsible for *serving* the lakeshore marts. Each team attempts to optimize their activity, for example find patterns of *ingestion*.

Though this model provides some level of scale, by assigning teams to different activities of the flow, it has an inherent limitation that does not scale what matters: delivery of outcome - in this case, delivery of new quality and trust-worthy data. Delivering an outcome demands synchronization between teams and aligning changes to the activities. Such decomposition is *orthogonal to the axis of change or outcome*, and slows down the delivery of value and introduces organizational friction.

Conversely, an outcome-oriented team decomposition, optimized for achieving an end to end outcome fast with low synchronization overhead.

Let's look at an example. Daff Inc. started its services with 'songs' and 'albums', and then extended to 'music events', 'podcasts', and 'radio shows'. Enabling a single new feature, such as visibility to the 'podcasts play rate', requires a change in all components of the pipeline. Teams must introduce new ingestion services, new cleansing and preparation as well as served aggregates for viewing podcast play rates. This requires synchronization across implementation of different components and release management across teams. Many data platforms provide generic and configuration-based ingestion services that can cope with extensions such as adding new sources easily or modifying the existing sources to minimize the overhead of introducing new sources. However this does not remove an end to end dependency management of introducing new datasets from the consumer point of view. The smallest unit that must change to cater for a new functionality, unlocking a new dataset and making it available for new or existing consumption, remains to be the whole pipeline - the monolith. This limits our ability to achieve higher velocity and scale in response to new consumers or sources of the data.

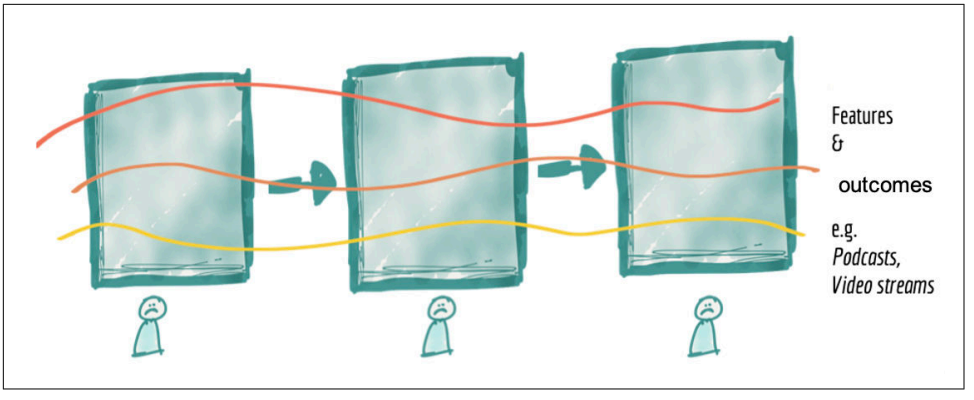

*Figure 3-10. Architecture decomposition is **orthogonal to the axis of change (outcome)** when introducing or enhancing features, leading to coupling and slower delivery*

We have created an architecture and organization structure that does not scale and does not deliver the promised value of creating a data-driven organization.

Recap

> The definition of insanity is doing the same thing over and over again, but expecting different results.
>
> —Albert Einstein

You made it, walking with me through the evolution of analytical data management architecture. We looked at the current state of the two-plane division between operational data and analytical data, and their fragile ETL-based integration model. We dug deeper into the limitations of analytical data management; limitations to scale - organizational scale in expansion of ubiquitous data, scale in diversity of usage patterns, scale in dynamic topology of data and need for rapid response to change. We looked critically into the root causes of their limitations.

The angle we explored was architecture and its impact on the organization. We explored the evolution of analytical data architectures from data warehousing, data lake to multi-modal warehouse and lake on the cloud. While acknowledging the evolutionary improvement of each architecture, we challenged some of the fundamental characteristics that all these architectures share: monolithic, centralized and technology driven. These characteristics are driven from an age-old assumption that to satisfy the analytical use cases, data must be extracted from domains, and consolidated and integrated under central repositories of a warehouse or a lake. This assumption was valid when the use cases of data were limited to low-frequency reports; it was valid when data was being sourced from a handful of systems. It is no longer valid when data gets sources from hundreds of microservices, millions of devices, from

within and outside of enterprises. It is no longer valid that use cases for data tomorrow are beyond our imagination today.

We made it to the end of Part I. With an understanding of the current landscape and expectations of the future, let's move to Part II and unpack what Data Mesh is based on its core principles.

PART II

What is Data Mesh

> "*... the only simplicity to be trusted is the simplicity to be found on the far side of complexity.*"
> —Alfred North Whitehead (Mathematician and Philosopher)

Data Mesh is a sociotechnical approach to share, access and manage analytical data in complex and large-scale environments - within or across organizations.

Since the introduction of Data Mesh in my original post, kindly hosted by Martin Fowler, I have noticed that people have struggled to classify the concept: Is Data Mesh an architecture? Is it a list of principles and values? Is it an operating model? After all, we rely on *classification of patterns*[1] as a major cognitive function to understand the structure of our world. Hence, I have decided to classify Data Mesh as a *sociotechnical* paradigm. An approach that recognizes the interactions between people and the technical architecture and solutions in complex organizations. An approach to data management that not only optimizes for the technical excellence of delivering analytical data sharing solutions but also improves the experience of all people involved, data providers, consumers and owners.

Data Mesh can be utilized as an element of an enterprise *data strategy,* articulating the target state of both the *enterprise architecture*, as well as an *organizational operating model* with an iterative execution model. Part III of this book focuses on its logical architecture and Part IV focuses on the operating model and the execution.

1 Hawkins, Jeff; Blakeslee, Sandra. On Intelligence (p. 165). Henry Holt and Co.

To get started with the understanding of Data Mesh, in this part of the book, I focus on its foundational principles. These principles are propositions and values that guide the behavior, structure and evolution of many Data Mesh implementations, yet to come. My intention for this part of the book is to create a baseline that we all agree upon, which then provides a foundation for the industry as a whole to create the practices, implementations and technology to bring Data Mesh to life. The purpose of the principles is to guide us all as we mature the evolution of our Data Mesh implementations.

This book is being written at the time that Data Mesh is arguably still in the *innovator, early adopter* phase of an innovation adoption curve.[2] It's at a phase that the venturesome *innovators* have embraced it and are already in the process of creating tools and technologies around it, and the highly respected *early adopters* have demonstrated shifting their data strategy and architecture inspired by Data Mesh. In fact in early 2020 InfoQ recognized Data Mesh as one of the *innovator* phase architectural trends. Hence, it's only appropriate to include the articulation of its principles and architectural style in my explanation of Data Mesh at this point in time, and leave the specific implementation details and tools to you to refine and build over time. I anticipate that any specific implementation design or tooling suggestion will be simply outdated by the time you get to read this book.

The Principles

There are four simple principles that can capture what underpins the logical architecture and operating model of Data Mesh. These principles are designed to progress us toward the objectives of Data Mesh: Increase getting value from data at scale, sustain agility as an organization grows, and embrace change in a complex and volatile business context. These goals and their relationship to the principles were introduced in <Chapter 2, After the Inflection Point>.

Here is a quick summary of the principles, and their motivation to achieve the Data Mesh values:

Domain-oriented ownership

Definition

Decentralize the ownership of sharing analytical data to business domains who are closest to the data — either are the source of the data or its main consumers. Decompose the data artefacts (data, code, metadata, policies) - logically - based on the business domain they represent and manage their life

2 Rogers, Everett (16 August 2003). Diffusion of Innovations, 5th Edition. Simon and Schuster. ISBN 978-0-7432-5823-4.

cycle independently. Essentially, align business, technology and analytical data.

Motivations

- Ability to scale out data sharing and consumption aligned with the axes of organizational growth: increased number of data sources, increased number of data consumers, increased diversity of data use cases and mode of access to data.
- Optimize for continuous change by localizing change to the business domains.
- Enable agility by reducing cross-domain synchronizations by removing centralized bottlenecks of data teams and warehouse or lake architecture.
- Increase data business truthfulness by closing the gap between the real origin of the data, and where and when it is shared for analytical use cases.
- Increase resiliency of the analytics and ML solutions by removing complex data pipelines.

Data as a Product

Definition

- Existing or new business domains become accountable to share their data as a product served to data users – data analysts and data scientists.
- Data as a product adhere to a set of usability characteristics such as below, among others:
  - — Guarantee a list of explicitly defined data quality metrics
  - — Provide ease of usability and understability
  - — Be accessible through a diverse set of methods, native to the tools of a spectrum of data users — analysts and scientists.
  - — Be Interoperabile and joinable with other domains' data products.

  Data as a product introduces a new unit of logical architecture called, data product quantum, controlling and encapsulating all the structural components — data, code, policy and infrastructure dependencies — needed to share data as a product autonomously.

Motivations

- Remove the possibility of creating low quality, untrustworthy domain-oriented data silos.
- Enable a data-driven innovation culture, by streamlining the experience of discovering and accessing high quality data — peer-to-peer — without friction.

- Enable autonomy through development of clear contracts between domains' data products with operational isolation — i.e. changing one shall not destabilize others.
- Getting higher value from data by sharing and using data easily across organizational boundaries.

Self-serve Data Platform

Definition

A new generation of self-serve data platform to empower domain-oriented teams to manage the end-to-end life cycle of their data products, to manage a reliable mesh of interconnected data products and share the mesh's emergent knowledge graph and lineage, and to streamline the experience of data consumers to discover, access, and use the data products.

Motivations

- Reduce the total cost of ownership of data in a decentralized domain-oriented operating model and architecture.
- Abstract data management complexity and reduce cognitive load of domain teams in managing the end-to-end life cycle of their data products.
- Enable a larger population of developers — generalist experts — to embark on data product development and reduce the need for specialization.
- Provide computational capabilities needed by the governance, such as automatically exercising policies at the right point of time — discovering data products, accessing a data product, building or deploying a data product.

Federated Computational Governance

Definition

A data governance operational model that is based on a federated decision making and accountability structure, with a team made up of domains, data platform, and subject matter experts — legal, compliance, security, etc. It creates an incentive and accountability structure that balances the autonomy and agility of domains, while respecting the global conformance, interoperability and security of the mesh. The governance model heavily relies on codifying and automated execution of policies at a fine-grained level, for each and every data product.

Motivations

- Ability to get higher-order value from aggregation and correlation of independent data products when the mesh behaves as an ecosystem following global interoperability standards.

- Counter the possible undesirable consequences of domain-oriented decentralizations: incompatibility and disconnection of domains.
- Make it feasible to achieve the governance requirements such as security, privacy, legal compliance, etc. across a mesh of distributed data products.
- Reduce the organizational overhead of continuously synchronizing between domains and the governance function, and to sustain agility in the face of continuous change and growth.

I expect the practices, technologies and implementations of these principles vary and mature over time, and for these principles remain unchanged.

I have intended for the four principles to be *collectively necessary and sufficient*; they complement each other and each address new challenges that may arise from others. Figure II-1 following diagram shows how these principles complement each other, and their main dependencies. For example, a decentralized domain-oriented ownership of data can result in data siloing within domains, and this can be addressed by the data as a product principle which demands domains to have an organizational responsibility to share their data with product-like qualities outside of their domain.

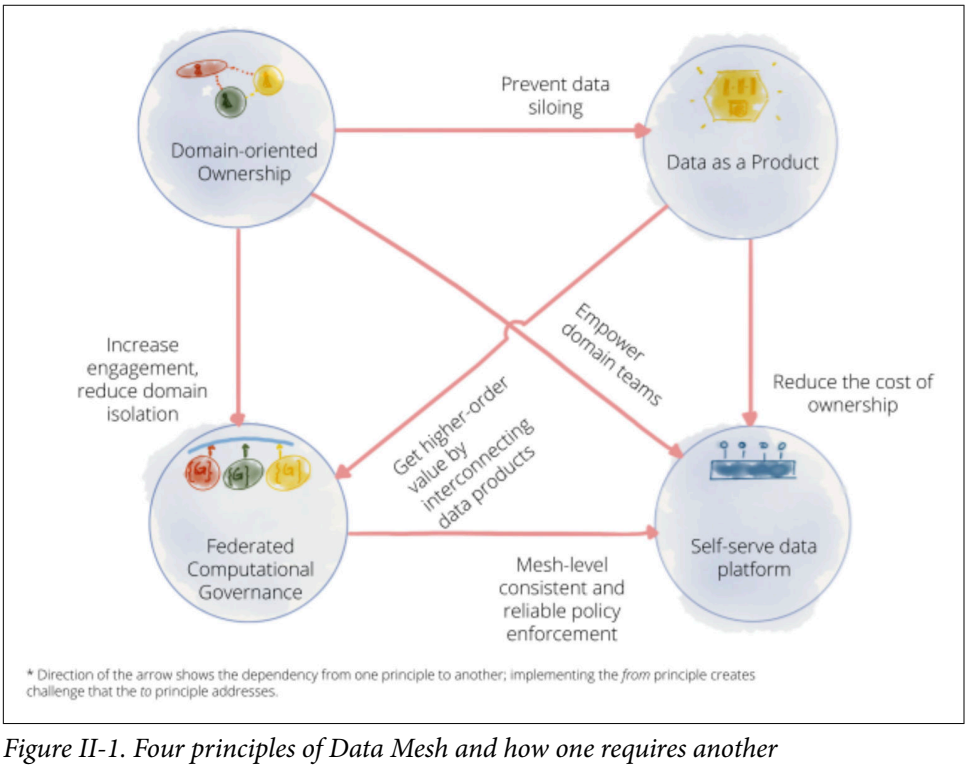

Figure II-1. Four principles of Data Mesh and how one requires another

Operationally you can imagine the principles in action as demonstrated in Figure II-2:

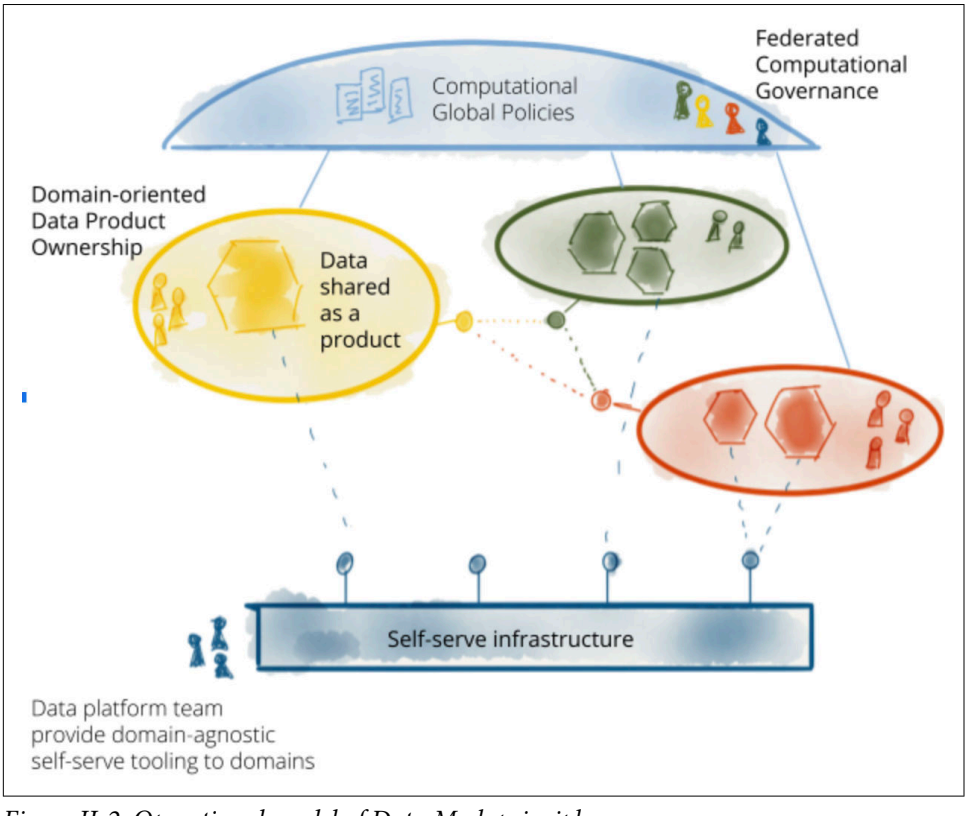

Figure II-2. Operational model of Data Mesh principles

The Origin

> *"To reject one paradigm without simultaneously substituting another is to reject science itself."*
>
> —Thomas S. Kuhn, The Structure of Scientific Revolutions

Thomas Kuhn, an American historian and philosopher of science, introduced the concept of *Paradigm Shift* in his rather controversial book 'The Structure of Scientific Revolutions (1962)'. He observed how science progressed in two main modes: incremental and revolutionary; that science progressed through long stretches of legato *normal science* where the existing theories form the foundation of all further research, followed with the occasional disruption of staccato *paradigm shifts* that challenged and transcended the existing knowledge and norm. For example, the paradigm shift of science from *Newtonian mechanics* to *Quantum mechanics,* when scientists attempt

to explain the governing laws of physics at the quantum level. Kuhn recognized that a prerequisite for *a paradigm shift* is identifying *anomalies*, observations that don't fit the existing norm, and entering the phase of *ciris,* questioning the validity of the existing paradigm in solving the new problems and observations. He also observed that people try, with increasing desperation, to introduce unsustainable complexities into the existing solutions to account for anomalies.

This almost perfectly fits the origin of Data Mesh and its principles. It came from the recognition of anomalies - failure modes and accidental complexities that I described Part I - and moments of crisis where the characteristics of the existing data solutions didn't quite fit the realities of enterprises today. We are in a moment of Khunian crisis. But it's only responsible to introduce a new set of principles for the new paradigm, the principles underpinning the premise of Data Mesh.

I wish I could claim that these principles were novel and new, and I cleverly came up with them. On the contrary, the principles of Data Mesh are generalization and adaptation of practices that have evolved over the last two decades and proved to address our last scale challenge: scale of *organization digitization*. These principles are the foundation of how digital organizations have solved organizational growth and complexity, while delivering unprecedented digital aspirations: moving all of their services to web, use mobile for every single touchpoint with their customers, and reduce organizational synchronizations through automation of most activities, etc. These principles are adaptation of what has enabled the Microservices[3] and APIs revolution and created platform-based team topologies[4], with computational governance models such as Zero Trust Architecture[5] to assure digital services operate securely in a distributed model across multiple clouds. In the last several years, I have been refining these principles and adapting them to analytical data problems.

Let's deep dive into each of them.

---

3 https://martinfowler.com/articles/microservices.html

4 Skelton, M., & Pais, M. (2019). Team topologies: organizing business and technology teams for fast flow. IT Revolution.

5 Rose, S. , Borchert, O. , Mitchell, S. and Connelly, S. (2020), Zero Trust Architecture, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-207, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420 (Accessed July 25, 2021)

CHAPTER 4

Principle of Domain ownership

A note for Early Release readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

Data mesh, at its core, is founded in *decentralization* and *distribution of responsibility* to people who are closest to the data in order to support continuous change and scalability. The question is, how do we decentralize and decompose the components of the data ecosystem and their ownership. The components include the *analytical data*, its associated information that makes the data usable and understandable (aka *metadata)*, the *code* that maintains the data and its integrity, and the *computation and infrastructure* necessary to create and serve it.

Contrary to the existing data architectures, Data Mesh follows the seams of organizational units as the *axis of decomposing data*. It does not follow the borders set by the underlying technology solutions, such as lake or warehouse implementations, nor the functional lines such as the centralized data or analytics team.

Our organizations today are decomposed based on their business domains. Such decomposition localizes the impact of continuous change and evolution - for the most part - to the domain. As you saw in <Chapter 3, Before the Inflection Point>, traditional data architectures are partitioned around technology, e.g. data warehouse,

and give data ownership to teams performing activities related to the technology, e.g. data warehouse team. The traditional architectures mirror an organizational structure that centralizes the responsibility of sharing analytical data to a data team, which has originally been setup to localize complexity of dealing with the new field of analytical data management. Data Mesh partitions data around business domains, e.g. podcasts, and gives data ownership to the domains, e.g. podcasts team. Data Mesh gives the data sharing responsibility to those who are most intimately familiar with the data, are first-class users of the data and are already in control of its point of origin.

This rather intuitive division of responsibility solves one problem but introduces others: it leads to a distributed logical data architecture, creates challenges around data interoperability and connectivity between domains. I will address these challenges with the introduction of the other principles in the upcoming chapters.

In this section, I unpack how to apply Domain-Driven-Design (DDD) strategies to decompose data and its ownership and introduce the transformational changes organizations need to put in place.

Apply DDD's Strategic Design to Data

Eric Evans's book Domain-Driven Design (2003) has deeply influenced modern architectural thinking and, consequently organizational modeling. It has influenced the microservices architecture by decomposing the systems into distributed services built around business domain capabilities. It has fundamentally changed how the teams form, so that a team can independently and autonomously own a domain capability.

Though we have adopted domain-oriented decomposition and ownership when implementing operational systems, curiously, ideas of decomposition based on business domains have yet to penetrate the analytical data space. The closest application of DDD in data platform architecture, I have seen, is for source operational systems to emit their business domain events and for the monolithic data platform to ingest them. However beyond the point of ingestion the domain teams' responsibility ends, and data responsibilities are transferred to the data team. The further transformations are performed by the data team the more removed the data becomes from its original form and intention. For example, in Daff Inc.'s podcasts' domain emits logs of podcasts being played on a short-retention log. Then downstream, a centralized data team will pick these events up and attempt to transform, aggregate, and store them as long-lived files or tables.

In his original book Eric Evans introduces a set of complementary strategies to scale *modeling* at the enterprise level called DDD's Strategic Design. These strategies are designed for organizations with complex domains and many teams. DDD's Strategic

Design techniques move away from the two widely used modes of modeling and ownership: (1) organizational-level central modeling or (2) siloes of internal models with limited integration, causing cumbersome inter-team communications. Eric Evan observed that total unification of the domain models of the organization into one, is neither feasible nor cost-effective. This is similar to data warehouse modeling. Instead, DDD's Strategic Design embraces modeling the organization based on multiple models each contextualized to a particular domain, called **Bounded Context**, with clear boundaries and, most importantly, with the articulation of the relationship between Bounded Contexts with **Context Mapping**.

Data Mesh, similarly to modern distributed operational systems and teams, adopts the boundary of Bounded Contexts to distribute the *modeling of analytical data*, the *data* itself and its *ownership*. For example, suppose an organization has already adopted a microservices or domain-oriented architecture and has organized its teams around domains' bounded contexts. In this case, Data Mesh simply extends this by including the analytical data responsibilities in the existing domains. This is the foundation of *scale* in a complex system like enterprises today.

Consider Daff Inc. for example, there is a 'media player' team who is responsible for the mobile and web digital media players. The 'media player' application emits 'play events' that show how listeners interact with the player. The data can be used for many downstream use cases, from improving the player applications performance to reconstructing the 'listener's session' and the longitudinal journey of discovering and listening to music. In the absence of a Data Mesh implementation, the 'media player' team basically dumps the play events - with whatever quality and cadence - on some sort of a short-retention streaming infrastructure or worse in its transactional database, which is then get picked up by the centralized data team to put into a lake or warehouse or likely both. This changes with Data Mesh. Data Mesh extends the responsibility of the 'media player' team to provide high-quality long-retention analytical view of the 'play events' — real-time and aggregated. The 'media player' team now has the end-to-end responsibility of sharing the 'play event' analytical data directly to the data analysts, data scientists or other persons who are interested in the data. The 'play events' analytical data is then transformed by the 'listener session' domain to get aggregated into a journey-based view of the listener interactions. The 'recommendations' domain uses the 'listener sessions' to create new datasets —graphs to recommend music based on listeners' social network's play behavior.

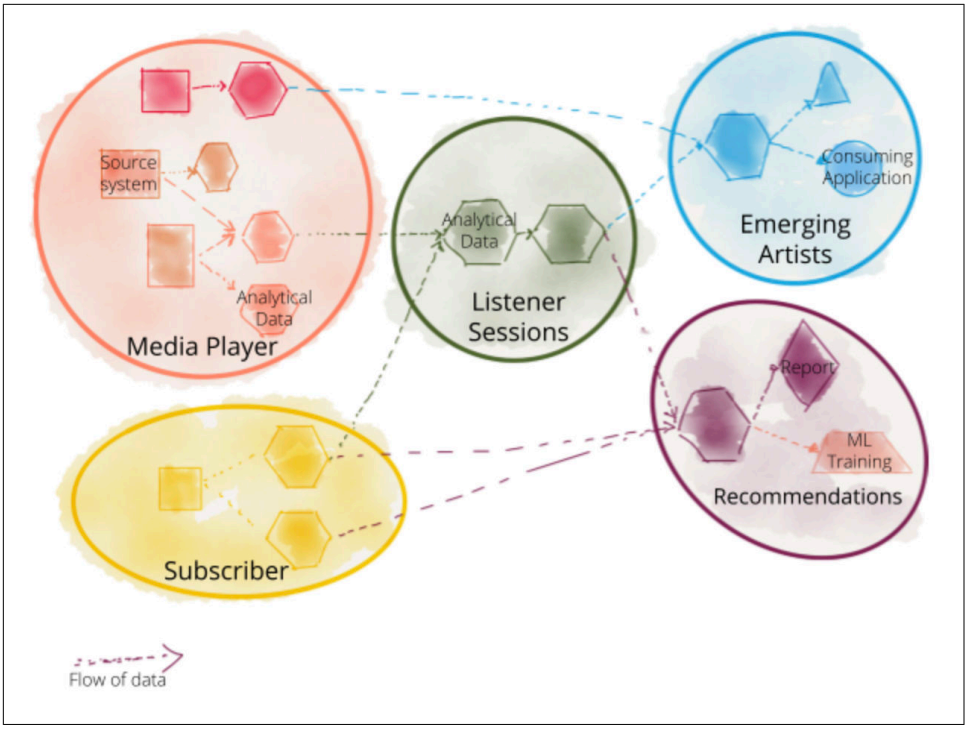

Figure 4-1. Decomposing the ownership and architecture of analytical data, aligned with the existing or new business domains.

Domain Data Archetypes

When we map the Data Mesh to an organization and its domains, we discover a few different archetypes of domain-oriented analytical data. While Data Mesh principles don't differentiate between the archetypes of domains, at a practical level, recognizing their characteristics help with optimizing the implementation and planning of the execution.

There are three archetypes of domain-oriented data:

- *Source-aligned domain data:* analytical data reflecting the business facts generated by the operational systems. This is also called *native* data product.
- *Aggregate domain data:* analytical data that is an aggregate of multiple upstream domains.
- *Consumer-aligned domain data:* analytical data transformed to fit the needs of one or multiple specific use cases and consuming applications. This is also called *fit-for-purpose* domain data.

Figure 4-2 expands on our previous example and demonstrates the archetypes of the domain-oriented data. For example, the 'media player' domain serves *source-aligned* analytical data directly correlated with the 'media player' application events. The 'listener sessions' serve *aggregate* data products that transform and aggregate individual listener player events into constructed sessions of interaction. And, 'recommendations' domain data is a *fit-for-purpose* data that serves the specific needs of the recommender service.

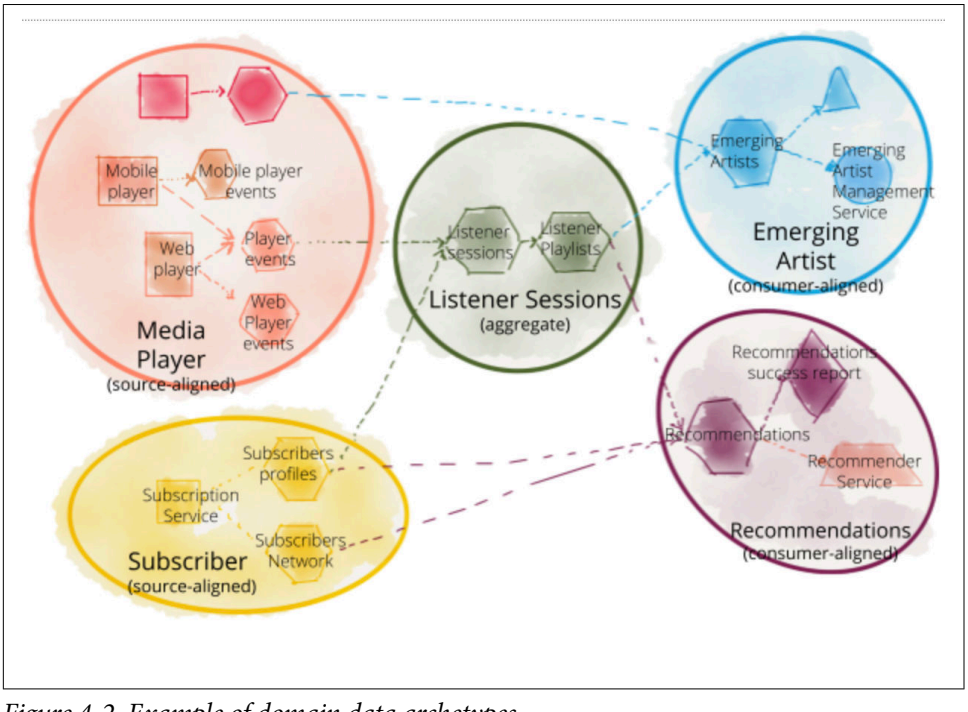

Figure 4-2. Example of domain data archetypes

Source-aligned Domain Data

Some domains naturally align with the source, where the data originates. The *source-aligned domain datasets* represent the *facts and reality of the business*. They capture the data that is mapped very closely to what the operational systems of their origin, *systems of reality*, generate. In our example, facts of the business such as 'how the users are interacting with the media players', or 'the process of onboarding labels' lead to the creation of source-aligned domain data such as 'user play events', 'audio play quality stream' and 'onboarded labels'. These facts are best known and generated by their corresponding operational systems. For example the 'media player' system knows best about the 'user play events'.

In summary, Data Mesh assumes that organizational units are responsible for providing business capabilities and responsible for providing the *truths of their business domain* as source-aligned domain datasets.

The business facts are best presented as business Domain Events, and can be stored and served as distributed logs of time-stamped events for any authorized consumer to access. In addition to timed events, source-aligned domain data should also be provided in easily consumable historical slices, aggregated over a time interval that closely reflects the interval of change in the business domain. For example in an 'onboarded labels' source-aligned domain, the monthly aggregation is a reasonable view to provide, 'onboarded labels in month <m>', processed and shared on a monthly basis.

Note that, source-aligned business events are not modeled or structured like the source application's transactional database; an anti-pattern is often observed, particularly when events are sourced through Change Data Capture tooling or Data Virtualization on top of the application's database. The application database model serves a very different purpose, and is often modeled for speed of performing transactions as needed by the application. The analytical data is structured for ease of understanding and access for reporting, machine learning training and other non-transactional use cases.

The nature of the domain data—*data on the outside*—is very different from the internal data that the operational systems use to do their job - *data on the inside*. Analytical data often has much larger volume and represents immutable timestamped information over an infinite[1] time. Its modeling likely changes less frequently than the source system, and its underlying storage and access technology should satisfy data access and querying for reporting and machine learning training, efficiently over a large body of data.

Source-aligned domain data is the most foundational. It is expected to be permanently captured and made available. As the organization evolves, its *data-driven* and *intelligence* services can always go back to the business facts, and create new aggregations or projections.

Note that source-aligned domain datasets closely represent the *raw data* at the point of creation, and are not fitted or modeled for a particular consumer. The archetypes introduced next cover the aggregations or the further transformations that might be needed for specific consumptions.

1 Infinite time captures data from a particular epoch to an unbounded future. Epoch may vary from system to system, depending on the retention policies of the domain. However it is expected to be long term.

Aggregate Domain Data

There is never a one to one mapping between a domain concept and a source system at an enterprise scale. There are often many systems that can serve parts of the data that belong to a shared concept—some legacy systems, some half modernized and some fresh greenfield ones. Hence there might be many *source-aligned data* aka *reality data* that ultimately need to be aggregated into a more aggregate view of a concept. For example, attributes that define 'subscribers', 'songs', or 'artists' can be mapped from many different points of origin. For example, the 'subscriber management' domain can have profile-related information about the subscribers, while the 'player' domain knows about their music preferences. There are use cases, such as marketing or sales, that demand a holistic view of the subscriber. This demands a new long-standing aggregate domain to be created with the responsibility of consuming and composing data from multiple source-aligned domains and sharing the aggregate data.

I strongly caution you against creating *ambitious aggregate domain data*; aggregate domain data that attempts to capture all facets of a particular concept like 'customer 360'. Such aggregate can become too complex and unwieldy to manage and difficult to understand and use for any particular use case. In the past, the discipline of Master Data Management (MDM) has attempted to aggregate all facets of shared data assets in one place and in one model. MDM suffers from the complexity and out-of-date challenges of a single unified cross-organizational canonical modeling.

Consumer-aligned Domain Data

Some domains align closely with the consuming use cases. The consumer-aligned domain data, and the teams who own them, aim to satisfy a closely related group of use cases. For example the 'social recommendation domain' that focuses on providing recommendations based on users' social connections, creates domain data that fit this specific need; perhaps through a 'graph representation of the social network of users'. While this graph data is useful for recommendation use cases, it might also be useful for other use cases such as a 'listeners notifications' domain. The 'listeners notification' domain provides data regarding different information sent to the listener, including what people in their social network are listening to.

Engineered *features* to train machine learning models often fall into this category. For example, Daff Inc. introduces a machine learning model that analyses the sentiment of a song, e.g. is positive or negative. Then uses this information for recommendation and ranking of their music. However, to perform sentiment analysis on a piece of music, data scientists need to extract a few features and additional information from the song such as 'liveliness', 'danceability', 'acousticness', 'valence', etc. Once these

attributes (features) are extracted, they can be maintained and shared as a consumer-aligned domain data to train the 'sentiment analysis' domain or other adjacent models such as 'playlist' creation.

The consumer-aligned domain data have a different nature in comparison to source-aligned domains data. They structurally go through more changes, and they transform the source domain events to structures and content that fit a particular use case.

I sometimes call these *fit-for-purpose* domain data.

Transition to Domain Ownership

Domain-oriented data ownership feels organic, and a natural progression of modern organizations' domain-driven digital journey. Despite that, it disputes some of the archaic rules of analytical data management. Below is a list of a few and I'm sure you can think of others.

Push Data Ownership Upstream

Data architecture nomenclature has flourished from the source of life itself: water. Data lake, Lakeshore marts, Dataflow, Lakehouse, data pipelines, lake hydration, etc. I do admit, it's a reassuring symbol, it's soothing and simply beautiful. However there is a dangerous notion lurking underneath it. The notion that data must flow from source to some other place - e.g. the centralized lake - to become useful, to become meaningful, to have value and to be worthy of consumption. There is an assumption that data upstream is less valuable or useful than data downstream.

Data Mesh challenges this assumption. Data can be consumable and useful right at the source domain, I called this source-aligned domain data. It doesn't need to flow from source to consumer through purifying pipelines, before it can be used. Data must be cleansed and made ready for consumption for analytical purposes upstream, by the source domain. The accountability for sharing quality analytical data is pushed upstream.

Of course at a later point downstream, source-aligned domain data can be aggregated and transformed to create a new higher order insight. I called these, aggregate domain data or fit-for-purpose domain data. This transformation happens within the context of downstream domains, under the domain's long-term ownership. There is no intelligent transformation that happens in the no man's land of in-between domains, in what is called a *data pipeline*, today.

Define Multiple Connected Models

Data warehousing techniques and central data governance teams have been in the search of the *one canonical model* holy grail. It's a wonderful idea, one model that

describes the data domains and can be used to provide meaning and data to all use cases. But in reality systems are complex, continuously changing and no one model can tame this messy life. Data Mesh in contrast follows DDD's Bounded Context and Context Mapping modeling of the data. Each domain can model their data according to their context, share these models and the corresponding data thoughtfully with others, and identify how one model can link and map to others.

This means there could be multiple models of the same concept in different domains and that is OK. For example, the 'artist' representation in the 'payment' includes payment attributes, which is very different from the 'artist' model in the 'recommendation' domain, that includes artist profile and genre. But the mesh should allow to map 'artist' from one domain to another, and be able to link artist data from one domain to the other. There are multiple ways to achieve this, including a unified identification scheme, a single ID used by all domains that include an 'artist'.

Polysemes, shared concepts across different domains, create linked data and models across domains.

Embrace the Most Relevant Domain, and Don't Expect the Single Source of Truth

Another myth is that we shall have a single source of truth for each concept or entity. For example, one source of truth to know everything about 'subscribers' or 'playlists', etc. This is a wonderful idea, and is placed to prevent multiple copies of out-of-date and untrustworthy data. But in reality it's proved costly, an impediment to scale and speed, or simply unachievable. Data Mesh does not enforce the idea of one source of truth. However, it places multiple practices in place that reduces the likelihood of multiple copies of out-of-date data.

The long-term domain-oriented ownership and accountability for providing discoverable, easily usable and of high quality, suitable for a wide range of users — analysts and scientists — reduces the need for copying.

Data Mesh endorses multiple models of the data, and hence data can be read from one domain, transformed and stored by another domain. For example, 'emerging artists' domain reads 'play events' domain data and transforms and then stores it as 'emerging artists'. This mimics the real world, data moves around, gets copied and gets reshaped. It's very difficult to maintain the ideology of a single source of truth under such a dynamic topology. Data Mesh embraces such dynamism for scale and speed. However, it continuously observes the mesh and prevents errors that often arise when data gets copied. Data Mesh prevents these errors through a set of complementary non-negotiable capabilities of the mesh and data shared on the mesh: immutability, time-variance, read-only access, discoverability and recognition of polysemes' unified identities cross-domain multiple domains. See <Chapter 9, Data Product Logical Architecture> for more on these.

Hide the Data Pipelines as Domains' Internal Implementation

The need for cleansing, preparing, aggregating and sharing data remains, so does the usage of *data pipeline*, regardless of using a centralized data architecture or Data Mesh. The difference is that in traditional data architectures, *data pipelines* are first-class architectural concerns that integrate to compose more complex data transformation and movement. In Data Mesh, a data pipeline is simply an internal implementation of the data domain and is handled internally within the domain. It's an implementation detail that must be abstracted from outside of the domain. As a result, when transitioning to Data Mesh, you will be redistributing different pipelines and their jobs to different domains.

For example the source-aligned domains need to include the cleansing, deduplicating, enriching of their domain events so that they can be consumed by other domains, without replication of cleansing downstream. Each domain data must establish a *Service Level Objective* for the quality of the data it provides: timeliness, accuracy, etc.

Consider Daff: our 'media player' domain providing audio 'play events' can include cleansing and standardization data pipeline in their domain that provides a stream of de-duped near-real-time 'play events' that conform to the organization's standards of encoding events.

Equally, we will see that aggregation stages of a centralized pipeline move into implementation details of aggregate or fit-for-purpose domain data.

One might argue that this model leads to duplicated effort in each domain to create their own data processing pipeline implementation, technology stack and tooling. Data Mesh addresses this concern with the self-serve data platform, described in <Chapter 6>. Having said that, domains are taking on additional responsibilities, the responsibilities and efforts shift from a centralized data team to domains, to gain agility and authenticity.

Recap

Arranging data and its ownership around technology has been an impediment to scale, it simply has been orthogonal to how change happens and features develop. Centrally-organized data teams have been the source of friction. There is an alternative, the alternative is a tried and tested method to scale modelling at enterprise level: Domain-oriented Bounded Context modeling. Data Mesh adapts this concept to the world of analytical data. It demands domain teams who are closest to the data to own the analytical data and serve the domain's analytical data to the rest of the organization. Data Mesh supports creation of new domain's data by composing, aggregating and projecting existing domains.

CHAPTER 5
Principle of Data as a Product

> **A note for Early Release readers**
>
> With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the 5th chapter of the final book.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

One of the long standing challenges of existing analytical data architectures is the high friction and cost of using data: *discovering, understanding, trusting, exploring, and ultimately consuming quality data*. There have been numerous surveys surfacing this friction. A recent report from Anaconda, a data science platform company, "The State of Data Science 2020", finds that nearly half of a data scientist's time is spent on data preparation - data loading and cleansing. If not addressed, this problem only exacerbates with Data Mesh, as the number of places and teams who provide data - i.e. domains - increases. Distribution of the organization's data ownership into the hands of the business domains raises important concerns around accessibility, usability and harmonization of distributed datasets. Further data siloing and regression of data usability are potential undesirable consequences of Data Mesh's first principle, *Domain-oriented ownership*. The principle of *data as a product* addresses these concerns.

The second principle of Data Mesh, *Data as a product,* applies *product thinking* to domain-oriented data, to not only remove such usability frictions but also truly

delight the experience of the data users - data scientists, data analysts, data explorers and anyone in between. *Data as a product* expects that the analytical data provided by the domains to be treated as a product, and the consumers of that data should be treated as customers - happy and delighted ones. Furthermore, data as a product underpins the case for Data Mesh, to unlock the value of an organisation's data by dramatically increasing the potential for serendipitous and intentional use.

Marty Cagan, a prominent thought leader in product development and management, in his book INSPIRED, provides convincing evidence on how successful products have three common characteristics: they are *feasible, valuable* and *usable. Data as a product* defines a new concept, called **data product** that embodies standardized characteristics to make data *valuable* and *usable.* Figure 5-1 demonstrates this point visually. This chapter introduces these characteristics. The next chapter, The Principle of Self-Serve Data Platform, describes how to make building data products *feasible.*

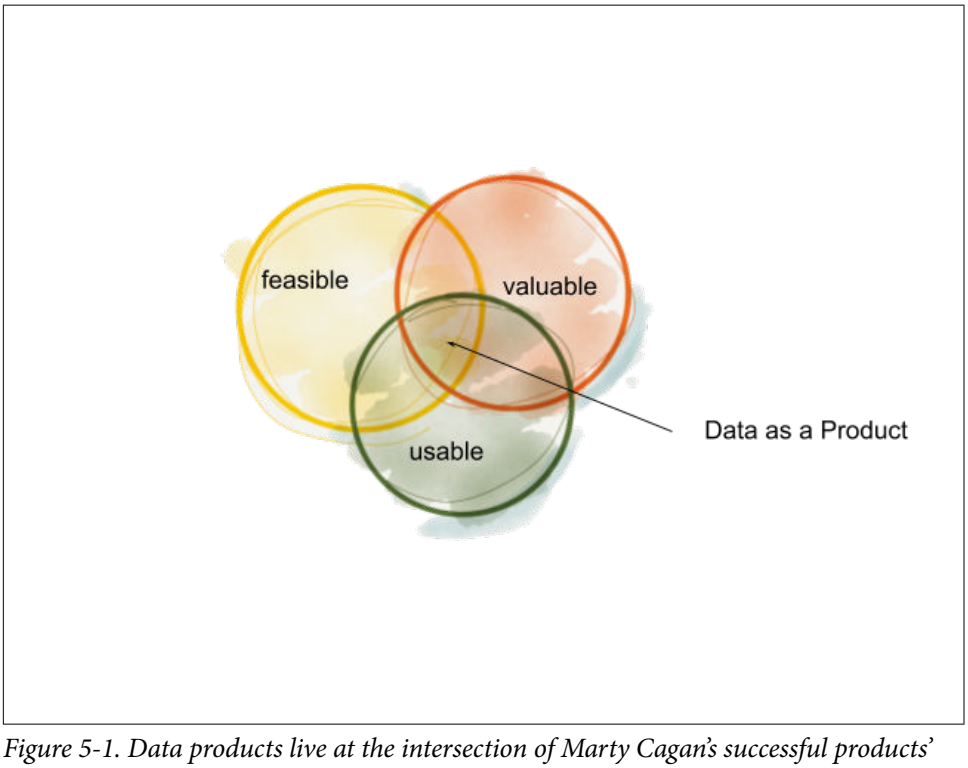

Figure 5-1. Data products live at the intersection of Marty Cagan's successful products' characteristics - INSPIRED

I must acknowledge that treating data as a product doesn't simply happen out of good intentions. For that, this principle introduces new roles to the domains such as *domain data product owner* and *data product developer* who have accountability and responsibility for creating, serving and evangelizing data as a product - meeting spe-

cific objective measures of data accessibility, quality, usability and availability. In chapter 12, The Organizational Design, I will cover further details about these new roles.

Data as a product inverts the model of responsibility compared to the past paradigms. In data lake or data warehousing architectures the accountability to create data with expected quality and integrity resides downstream from the source; in the case of data warehouse the accountability remains with the warehouse team, and in the case of the lake it's left with the consumers — recalling the 45% data scientist time spent on data cleansing, on a good day. Data Mesh shifts this responsibility left as close to the source of the data as possible. This transition is not unique to Data Mesh, in fact over the last decade we have seen the trend of *shift left* with testing and operations, on the basis that addressing problems is cheaper and more effective done close to the source.

I go as far as saying, *data* is not what gets shared on a mesh, it is only a *data product* that can be worthy of sharing on the mesh.

The concept of data ownership in this book is limited and scoped to the accountability of the organizations to maintain the quality, longevity and lawful accessibility of the data they generate in transaction with internal and external entities such as the users, customers, and other organizations. The ultimate sovereignty of such data remains with the users, customers, or other organizations whose data is being captured and managed. The organizations act as *data product owners* while the individuals remain the *data owners.*

The concept of self-sovereign data — individuals having the full control and authority over their personal data — is close and dear to my heart but outside of the scope of this work. I believe Data Mesh can establish the foundation toward self-sovereign data, but that would be the topic of a different book.

Of course to close the gap further between the desired state of having data as a product and a technical implementation, I need to introduce the architectural parallel of a data product. This architectural component is called a *data quantum* and I will introduce that in chapter 8, The Logical Architecture.

For now, let's unpack how we can apply product thinking to data.

Apply Product Thinking to Data

Over the last decade, high-performing organizations have embraced the idea of treating their internal operational technology like a product, similarly to their external technology; treating their internal developers as customers and their satisfaction a sign of success. At ThoughtWorks, we have noticed this trend particularly in two

areas: applying product management techniques to internal platforms which accelerates the internal developers to build and host solutions more easily on top of internal platforms, as well as treating APIs as a product to build APIs that are discoverable, understandable, easily testable to assure an optimal developer experience.

Applying the magic ingredient of product thinking to internal technology begins with establishing empathy with internal consumers i.e. fellow developers, and collaborating with them on designing the experience, gathering usage metrics, and continuously improving the internal technical solutions over time to maintain ease of use; building internal tools that are valuable to the developers and ultimately business.

Curiously, the magical ingredient of empathy, treating data as a product and its users as customers, has been missing in big data solutions. Operational teams still perceive their data as *a byproduct* of running the business, leaving it to someone else, e.g. the data analytics team, to pick it up and recycle it into *products*. In contrast, Data Mesh domain teams apply product thinking with similar rigor to the datasets that they provide, striving for the best data user experience.

Consider Daff Inc.'s example. One of its critical domains is the 'media player'. The 'Media player' domain provides essential data such as what songs have been played by whom, when, and where. There are a few different key data consumers for this information. For example, the 'media player support' team is interested in near-real-time events to catch errors causing a degrading customer experience quickly and recover the service, or respond to incoming customer support calls informedly. On the other hand, the 'media player design' team is interested in aggregated play events that tell a data story about the listener's journey over a longer period of time, to improve the media player features toward a more engaging listener experience.

In this case, the 'media player' domain provides two different datasets as its products to the rest of the organization; 'near-real-time play events' exposed as infinite event logs, and aggregated play events exposed as serialized files on an object store. The 'media player' domain needs to know their data customers and how they wish to access the data to continuously strive to improve their experience.

While you can adopt the majority of the product ownership techniques, there is something unique about data. The difference between data product ownership and other types of products lies in the *unbounded nature of data use cases*, the ways in which a particular data can be combined with other data and ultimately turned into insights and actions. At any point in time, data product owners are aware or can plan for what is known today as viable use cases for their data. While there remains a large portion of unknown future use cases for their data produced today, perhaps beyond the imagination of the data product owners. This is more true for source-aligned domains and less for consumer-aligned ones. To design data as a product that balances the immediate known use cases and the unknown ones, the source-aligned data product owners must strive to model the reality of the business, as closely as possible,

in their data. For example, capturing all the 'player events' as an infinite high resolution log, so that we can build other transformations and infer future insights from the data that is captured today.

Baseline usability characteristics of a data product

Individual domains must offer a set of domain-specific measurable characteristics that demonstrate the value and impact of their specific datasets as products. For example, the decision on resolution of 'player events' or how long the domain is going to retain those events, affect the usability of this specific domain.

In addition to that, there are a set of baseline characteristics that all data products regardless of their domain must exhibit. I call these *baseline data usability characteristics*. Every data product must exhibit these characteristics to be part of the mesh. Figure 5-2 lists the baseline data usability characteristics that all data products share, data products of all domains and all types of data.

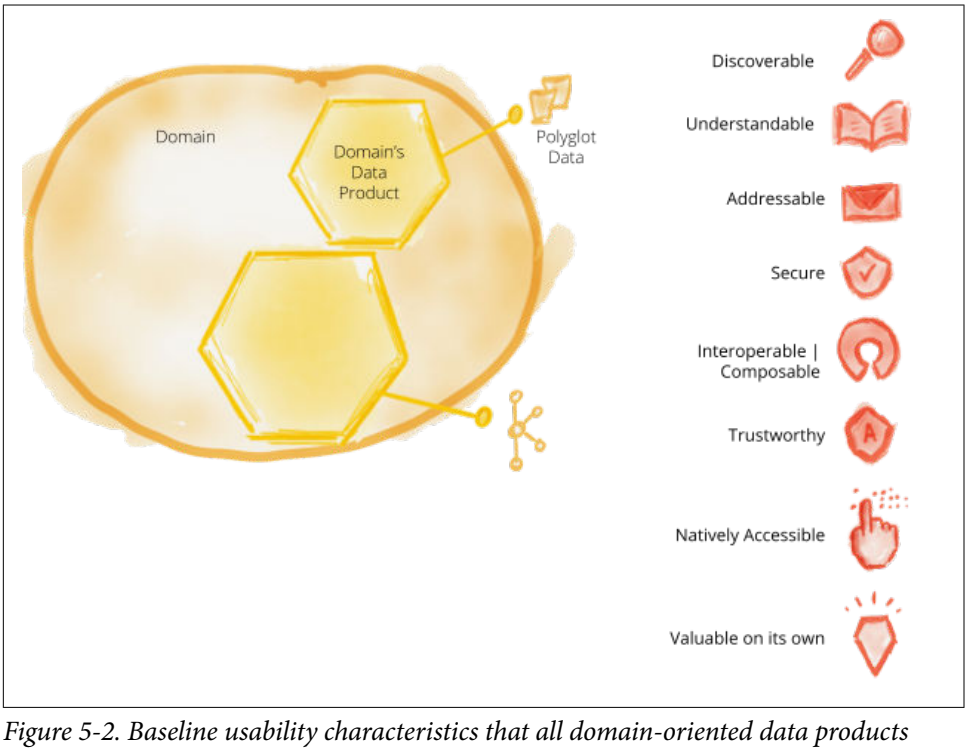

Figure 5-2. Baseline usability characteristics that all domain-oriented data products share

Note that these are *usability characteristics* and are oriented around the experience of the consumers. These are not intended to express the *technical characteristics* relevant

to the perspective of the data product developers. Chapter 9, Data Quantum Design, covers the underlying technical characteristics.

The baseline characteristics listed in this section are an addition to what we have known as FAIR data in the past — data that meet principles of *findability*, *accessibility*, *interoperability*, and *reusability*. FAIR principles were published in 2016 in *Scientific Data,* a peer-reviewed scientific journal. In addition to these principles I have introduced characteristics that are necessary to make distributed data ownership and architecture work.

Let's put ourselves in the shoes of data consumers and walk through the baseline data usability characteristics.

Discoverable

> "Two of the most important characteristics of good design are discoverability and understanding. Discoverability: figure out what actions are possible and where and how to perform them? Understanding: What does it all mean? How is it supposed to be used? What do all the different controls and settings mean?[1]
>
> —Don Norman (Cognitive Science and design researcher, professor, and author

The very first step that data users take in their journey, is to discover the world of data available to them, explore and search to find "the one". Hence one of the first usability attributes of data is to be easily discoverable. Data consumers — or data users, both of which I use interchangeably — need to be able to explore the available data products, search and find the desirable sets, explore them and gain confidence in using them. A traditional implementation of discoverability is often approached as some centralized registry or catalogue listing available datasets with some additional information about each dataset, the owners, the location, sample data, etc. Often this information is curated, after the fact that data is generated, and by the centralized data team or the members of the governance team.

Data product discoverability on Data Mesh embraces a source-oriented solution, where discoverability information is *intentionally* provided by the data product itself. Data Mesh embraces the dynamic topology of the mesh, continuously evolving data products, and the sheer scale of available data products. Hence it relies on individual data products to provide their discoverability information at different points of their lifecycle — build, deploy and run — in a standard way. Each data product continuously shares its source of origin, owners, run-time information such as timeliness, quality metrics, sample datasets, and most importantly information contributed by their consumers such as the top use cases and applications enabled by their data.

1 Norman, D. A. (2002). *The design of everyday things*. Basic Books.

Chapter 9, Data Quantum Design, will discuss the technical design of a data product discoverability.

Understandable

The next step in a data user's journey, once they've discovered a data product, is to understand it. Get to know the semantic of its underlying data, as well as various syntax in which the datasets are presented to the data user.

Each data product provides a semantically coherent dataset; a dataset with a specific *meaning*. A data user needs to understand this *meaning,* what kind of entities the data product encapsulates, what are the relationships among the entities and their adjacent data products. Back to our 'media player event' example, a data user should easily understand what constitutes a player event: the 'user', the 'play actions' they have taken and the 'time' and 'location' of their action and the 'feedback' the 'action' has resulted in. The data user should easily understand the kinds of 'actions' available and that there is a relationship between 'player event's listener' and that of a 'subscriber' from the adjacent 'users' domain. Data products must provide a formal representation of such *semantics*.

In addition to an understanding of the semantic, data users need to understand how exactly the data is presented to them. How it is serialized and how they can access and query the data syntactically; what kind of SQL queries can they execute or how they can read the data objects - understand the *schema* of the underlying representation of the data. This is ideally accompanied by sample datasets and example consumer codes.

Data schemas are just a starting point to support the understanding of the data product in a self-serve manner. Additionally, dynamic and computational documents like computational notebooks are great companions to tell the story of a data product. Computational notebooks include documentation of the data, as well as code to use it, with an immediate feedback of visually demonstrating the code in action.

Lastly, understanding is a social process. We learn from each other. Data products must facilitate communication across their consumers to share their experience and how they take advantage of the data product.

Understanding a data product requires no user hand holding. A self-serve method of understanding is a baseline usability characteristic.

Trustworthy and Truthful

> "Trust: a confident relationship with the unknown". [2]

2 https://medium.com/@rachelbotsman/trust-thinkers-72ec78ec3b59

—Rachel Botsman (Trust Fellow at Oxford University)

No one will use a product that they can't trust. So what does it really mean to trust a data product, and more importantly what does it take to trust? To unpack this, I like to use the definition of trust offered by Rachel Botsman: *the bridge between the known and the unknown*. A data product needs to close the gap between what data users know confidently about the data, and what they don't know but need to know to trust it. While the prior characteristics like understandability and discoverability close this gap to a degree, it takes a lot more to trust the data for use.

The data users need to confidently know that the data product is truthful - represents the fact of the business. They need to confidently know how closely data reflects the reality of the events that have occurred, the probability of truthfulness of the aggregations and projections that have been created from business facts.

Each data product needs to communicate and guarantee its **service level objectives** (SLOs)- objective measures that remove uncertainty surrounding the data. This includes measuring conformance to them, monitoring them and triggering support processes if SLOs are broken. The data product SLOs include, among others:

Interval of change
: How often changes in the data are reflected.

Timeliness
: The skew between the time that a business fact occurs and is served to the data users.

Completeness
: Degree of availability of all the necessary information,

Statistical shape of data
: Its distribution, range, volume, etc.

Lineage
: The data journey from source to now.

Precision and accuracy over time
: Degree of business truthfulness as time passes.

Operational qualities
: Freshness, general availability, performance, etc.

In the traditional data platforms it's common to extract and onboard data that has errors, or does not reflect the truth of the business and simply can't be trusted. This is where the majority of the efforts of centralized data pipelines are concentrated, cleansing data after ingestion.

In contrast, Data Mesh introduces a fundamental shift that the owners of the data products must communicate and guarantee an acceptable level of quality and trustworthiness - unique to their domain - as a component of their data product. This requires applying data cleansing and automated data integrity testing at the point of creation of the data product.

*P*roviding data provenance and data lineage — the data journey, where it has come from and how it got here — as the metadata associated with each data product helps consumers gain further confidence in the data product and its suitability for their particular needs. I'm of the opinion that once the discipline of building trustworthiness in each data product is established, there is less need for establishing trust through investigative processes and applying detective techniques navigating the lineage tree. Having said that, data lineage will remain an important element in a few scenarios, such as postmortem root cause analysis, debugging, and evaluation of data's fitness for ML training.

Addressable

A data product must offer a permanent and unique address to the data user to programmatically or manually access it. A unique addressing system must embrace the dynamic nature of the data and the mesh topology. It must embrace the aspects of data products that can continuously change while supporting *continuity of usage*. The addressing system must accommodate the following continuously changing aspects of a product, among others:

- Continuous change in the data product's semantic and syntax: schema evolution,
- Continuous release of new data time slices: partitioning strategy and grouping of data tuples associated with a particular time (duration),
- Newly supported syntaxes of the underlying data: new ways of serializing, presenting and querying the data,
- Continuously changing run-time behavioral information: e.g. SLOs.

The unique address must follow a global convention that helps the users to programmatically and consistently access all data products. The data product must have an addressable aggregate root that serves as an entry to all information about a data product, including its documentation, SLOs, the datasets it serves, etc. The data product address can be discovered through the *Discoverability* solution.

Interoperable and Composable

One of the main concerns in a distributed data architecture is the ability to correlate data across domains and stitch them together in wonderful and insightful ways: join, filter, aggregate. The key for an effective composability of data across domains is fol-

lowing standards and harmonization rules, that allow *linking data* across domains easily.

Here are a few things data products need to standardize to facilitate interoperability and composability:

Field type
: A common explicitly defined type system.

Polysemes Identifiers
: Universally identifying entities that cross boundaries of data products.

Data products global addresses
: A global unique address allocated to each data product, ideally with a uniform scheme for ease of establishing connections to different data products.

Common metadata fields
: Such as representation of time when data occurs and when data is recorded.

Schema linking
: Ability to reuse and link to schemas — types — defined by other data products.

Data linking
: Ability to link to data in other data products.

Schema stability
: Approach to evolving schemas that respects backward compatibility.

For example, let's look at managing polysemes identifiers. At Daff Inc. business, 'artist' is a core business concept that appears in different domains. An 'artist', while remaining to be the same global entity, has different attributes and likely different identifiers in each domain. The 'play event' data product may recognize the artist differently to the 'payment' domain that takes care of invoices and payments for artists royalties. In order to correlate data about an 'artist' across different domain data products we need to agree on how we identify an 'artist', globally, as a polyseme — an entity that crosses multiple domain boundaries.

Chapter 7, The Principle of Computational and Federated Governance covers the topic of global standards and protocols applied to each data product to create a happy ecosystem of interoperable data products. Interoperability is the foundation of any distributed system design and Data Mesh is no exception.

Secure

Data users must access the data product securely and in a confidentiality-respecting manner. This is a must, whether the architecture is centralized or distributed. However in the world of decentralized architecture like Data Mesh, the access control is validated by the data product, right in the flow of data, *access, read or write.* The

access control policies can change dynamically and continuously get evaluated at each point of access to the data product's data. Additionally, access to a data product is not quite binary — whether the user can see or can't see the data. In many cases while the user may not be able to actually see the record, it might have sufficient permissions to see the shape of the data and evaluate using the data given its statistical characteristics. Similarly to operational domains the access control policies can be defined centrally but enforced at run time by each individual data product. Data products must follow the practice of security policy as code. This is to write security policies in a way that can be versioned, computationally tested and enforced, can be deployed and observed.

A policy described, tested and maintained as code, can articulate various security related concerns such as the ones below, among others:

Access control
: Who, what and how systems can access the data product

Encryption
: What kinds of encryption - on disk, in memory, or in transit - using which encryption algorithm, how to manage keys and minimize the radius of impact in case of breaches

Confidentiality levels
: What kinds of confidential information e.g. personally identifiable information, personal health information - the data product carries

Data retention
: How long the information must be kept

Regulations and agreements
: GDPR, CCPA, domain-specific regulations, contractual agreements

Natively Accessible

Depending on the data-maturity of the organization there is a wide spectrum of data user personas in need of access to data products. The spectrum spans from data analysts that are comfortable with exploring data in sheets to the ones that create statistical models — visualizations or reports — of the data, to data scientists that curate and structure the data to train their models, or analytical application developers that expect a real-time stream of events. This is a fairly wide spectrum of users with equally diverse expectations of how to fetch and use data.

There is a direct link between the *usability* of a data product, and how easily a particular data user can access it with their native tools. For example, the 'play events' data product needs to natively support fetching data through a SQL query to satisfy the

native mode of access by a data analyst, as well as reading a stream of events, for a data scientist.

Valuable on Its Own

I think it's fairly obvious that a data product must be valuable — it should have some inherent value for the data users in service of the business and customers. After all if the data product owner can't envisage any value out of the data product, perhaps best not create one. Having said that, it's worth calling out that a data product should carry a dataset that is *valuable* and *meaningful* on its own — without being joined and correlated with other data products.

Of course, there is always higher order meaning, insight and value that can be derived from the correlation of multiple data products, but if a data product on its own serves no value it should not exist.

There is an important differentiation between units that constitute Data Mesh, i.e. data products, and units that constitute a data warehouse, tables. In the data warehouse modeling, there are often glue tables that optimize correlation between entities, i.e. identity tables that map identifiers of one kind of entity to another. Such identity tables are not meaningful or valuable on their own — without being joined to other tables — they are mechanical implementations to facilitate correlations. This is not true about data products. Data Mesh does not demand mechanical data products that simply facilitate joining other data products.

Summary

For data to be a product it must adhere to a set of rules and exhibit a set of traits that make it fit right in the intersection of Cargan's *usability, feasibility, and valuable* Venn diagram. For data to be a product, it must be valuable on its own, and in cooperation with other data products. It must demonstrate *empathy* for its users, be accountable for its usability and integrity.

Transition to Data as a Product

In working with my clients, I have found that they are overwhelmingly receptive to Data Mesh principles, often questioning "why I didn't think of it myself" or occasionally saying "we have been doing something similar but not quite the same". The principles appear to be intuitive and rather natural next steps in their organizations' tech modernization journey; an extension to modernization of the operational aspect of organizations - e.g. moving toward domain-oriented ownership of capabilities with microservices and internal product thinking treating operational APIs as products.

However, their sense of discomfort arises when we go deeper into what it actually takes to implement the transformation toward Data Mesh, going beyond words and

clearly contrasting the *system of the world* of Data Mesh against the past. What I found in my conversations with Data Mesh early implementers is that while they verbalize the principles and their intention to implement them, the implementations remain heavily influenced by the familiar techniques of the past.

For this reason, I have decided to include a number of thought-provoking *transition* statements as well as a few pragmatic steps to crystalize the differences between the existing paradigm and truly *serving and owning data as a product*.

I invite you to think of new transition statements that I likely have missed here.

Include Data Product Ownership in Domains

Movements such as DevOps — closing the gap between building and operating business services — as well as collaborative product-oriented teams have been moving the companies from *functional teams* — separate teams for each function of design, dev, ops — to *cross-functional teams* - integrated teams of design, dev, ops.

Introduction of analytical data as a product adds to the list of existing responsibilities of cross-functional domain teams, and expands their roles:

- *Data product developer:* the role responsible for developing, serving and maintaining the domain's data products as long as the data products remain to exist and serve its consumers.
- *Data product owner:* the role accountable for the success of domain's data products in delivering value, satisfying and growing the data consumers, and defining the lifecycle of the data products.

Figure 5-3 shows the sample of products that a domain creates and maintains — operational applications and data products. The domain team's cross-functional team includes new roles to maintain its data products.

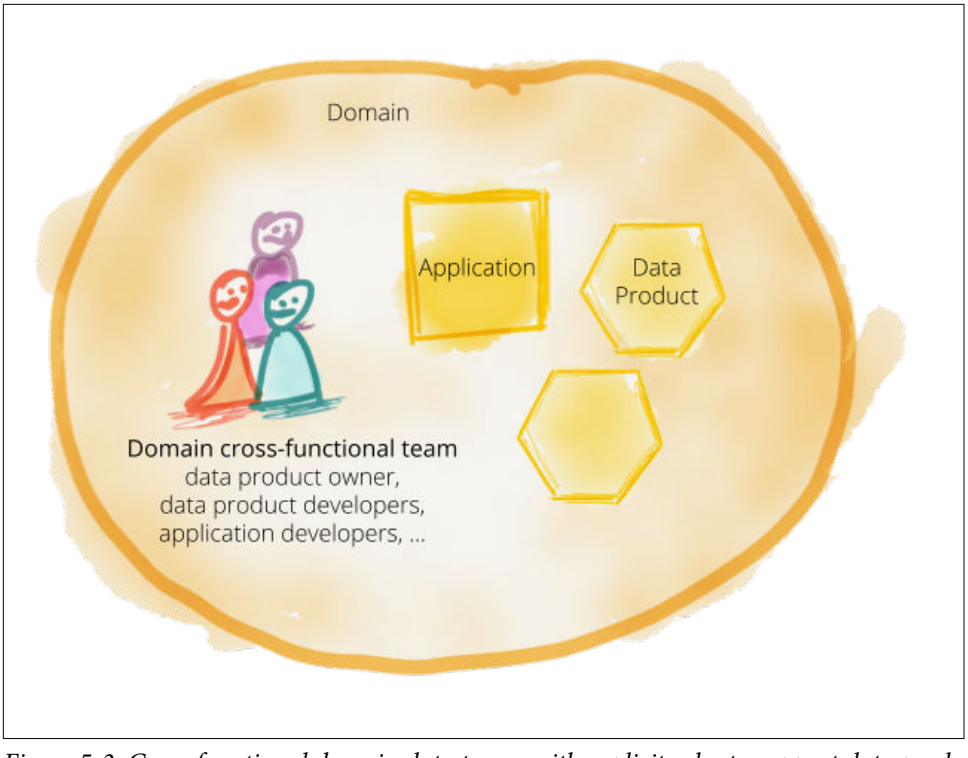

Figure 5-3. Cross-functional domain data teams with explicit roles to support data product ownership and development

Introduce the Domain Data Product Owner Role. Data product owners make decisions around the vision and the roadmap for the data products, concern themselves with the satisfaction of their data consumers, and continuously measure and improve the quality and richness of the data their domain owns and produces. They are responsible for the lifecycle of the domain datasets, when to change, revise and retire data and schemas. They strike a balance between the competing needs of the domain data consumers.

Data product owners must define success criteria and business-aligned Key Performance Indicators (KPIs) for their data products — a set of objective metrics that measures the performance and success of their data products. Like any product the success criteria must be aligned with the satisfaction of the data consumers. For example, *data consumer satisfaction* can be measured through net promoter score — the rate of consumers that recommend the data products to others, the decreased lead time for data product consumers to discover and use the data product successfully, and the growth of users.

Introduce the Domain Data Product Developer Role. In order to build and operate the domains' data products, the development team's roles need to be extended to include data product developers. Data product developers work closely with their collaborating application developers in defining the domain data semantic, mapping data from application context, data on the inside, to the data product context, data on the outside. They build and maintain the transformation logic that generates the desired data. They build and maintain the usability traits and guarantees of their data products.

A wonderful side effect of such a cross-functional team is cross pollination of different skill sets and ability to cross skill the existing developers to become data product developers.

Reframe the Nomenclature to Create Change

One of the commonly used phrases in data analytics is *ingestion*, receiving data from some upstream source — often an untrustworthy source that has *egested* data as a *byproduct* of its operation. It's now the job of the downstream pipeline is to ingest, cleans, process the data before it can be *consumed* to generate value.

Data Mesh suggests reframe receiving upstream data from *ingestion* to *consumption*. The subtle difference is that the upstream data is already cleansed, processed and *served* ready for consumption. The change of language creates new cognitive framing that is more aligned with the principle of serving data as a product.

Relatedly, the word *extraction* used in ETL (extract, transform, load) and its other variations, need to be critically evaluated. *Extraction* evokes a passive role for the provider, and an intrusive role for the consumer. As we know, extracting data from operational databases that are not optimized for use other than their application's create all kinds of pathological coupling and a fragile design. Instead, we can shift the language to *publish* or *serve*, and of course *consume* by the user. This implies shifting the implementation of data sharing from accessing raw databases, to intentionally sharing domain events or other interfaces.

By now you probably have picked up on my emphasis on language and metaphors we use. George Lakoff - professor of Cognitive Science and Linguistics at UC Berkeley - in his book, Metaphors we live by, elegantly demonstrates the consequence of shifting our language around the concept *argument,* from *war* to *dance*. Imagine the world we would live in and the relationships we would build, if instead of *winning* an argument, losing and gaining argument ground, and attacking the weak points of an argument we would, as dancers, perform a balanced and aesthetically pleasing argument, express our ideas and emotions through a beautiful and collaborative ritual of dancing.

Think of Data as a Product, Not a Mere Asset

"Data is an asset", "Data must be managed like an asset". These are the phrases that have dominated our vernacular in big data management.

The metaphor of *asset* used for *data* is nothing new. After all, for decades, TOGAF, a standard of The Open Group for Enterprise Architecture methodologies and frameworks explicitly has penciled in "Data is an Asset" as the first principle of its data principles. While on the surface this is a rather harmless metaphor, it has shaped our perceptions and actions toward negative consequences. For example, our actions toward how we measure success. Based on my observations Data as an asset has led to measuring success by *vanity metrics* — metrics that make us look or feel good but don't impact performance — such as the *number of datasets and tables* captured in the lake or warehouse, or the *volume of data*. These are the metrics I repeatedly come across in organizations. Data as an asset promotes keeping and storing data rather than sharing it. Though TOGAF's "Data is an asset" principle is immediately followed by "Data is shared".

I suggest the change of metaphor to *data as a product*, and a change of perspective that comes with that. For example, measuring success through *adoption of data*, its *number of users*, and their *satisfaction* using the data; underscoring *sharing the data* vs. keeping and locking it up and putting emphasis on the continuous care that a quality product deserves.

I invite you to spot other metaphors and vocabulary that we need to reshape to construct a new system of concepts for Data Mesh.

Establish a Trust-but-verify Data Culture

Data as a product principle implements a number of practices that lead to a culture where data consumers, by default, can trust the validity of the data, while verifying its fitness for their use cases.

These practical changes include: introducing the role for long-term ownership of a data product, accountable for the integrity, quality, availability and other usability characteristics of the data product; introducing the concept of a data product that not only shares data but also explicitly shares a set of objective measures such as timeliness, retention, and accuracy; creating a data product development process that automats testing of the data product transformation code as well as the integrity of the data it produces.

Today, in the absence of these data-as-a-product practices, *data lineage* remains a vital ingredient for establishing trust. The large gap between data providers and data consumers, data providers lack of visibility to the consumers and their needs, lack of long-term accountability of the actual data providers, and the absence of computa-

tional guarantees, have left consumers with no choice but to assume data is untrustworthy and requires a detective investigation through lineage before it can be trusted.

Data-as-a-product practices aim to build a new culture, from *presumption of guilt*, to the Russian proverb of *trust by verify.*

Join Data and Compute as One Logical Unit

Let's do a test. When you hear the word data product what comes to your mind? What shape? What does it contain? How does it feel? I can guarantee that a large portion of readers imagine static files or tables - columns and rows, some form of storage medium. It feels static, and perhaps of trustworthy quality. It's accumulated. It's content is made of bits and bytes that are representative of the facts, beautifully modeled. That is intuitive, after all by definition *datum* - singular form - is a "*piece of information*".

The result of this perspective is the separation of *code (compute)* from *data;* in this case, separation of the code that maintains the data, creates it and serves it. This separation creates orphaned datasets that overtime decay. At scale, we experience this separation as data swamps — a deteriorated data lake.

Data Mesh shifts from this dual mode of data vs. code to data and code as one architectural unit. A single deployable unit that is structurally complete to do its job, providing high-quality data of a particular domain. None exists without the other.

The data and code co-existence as one unit is not a new concept for people who have managed Microservices architecture. The evolution of operational systems has moved to a model that each service manages its code and data, its schema definition and upgrades. The difference between an operational system is the relationship between the *code* and its *data.* In the case of Microservices architecture, *data* serves the *code;* it maintains state so that code can complete its job, serving business capabilities. In the case of a data product and Data Mesh this relationship is inverse, *code* serves *data;* the transformation logic is there to create the data and ultimately serve it.

Note that the underlying physical infrastructures that host code and data can remain independent.

Interestingly, the original definition of *datum,* from 18th century Latin, is "something given". This original definition is much closer to the data product's spirit.

Recap

The mindshift, the principle, and the design of *Data as a product* is not only a response to the *data siloing* challenge that may arise from distribution of data ownership to domains, but also is a shift in the data culture toward *data accountability* and *data trust* at the point of origin. The ultimate goal is to make data simply *usable*.

We looked at eight non-negotiable baseline *usability* characteristics including *discoverability, understandability, trustworthiness*, *addressability, security, interoperability, native accessibility, and independently valuable.* We introduced the role of *data product owner* - someone with an intimate understanding of the domain's data and its consumers - to assure continuity of ownership of data and accountability of success metrics such as *data quality, decreased lead time* of data consumption, and in general *data user satisfaction* through net promoter score.

Each domain will include *data product developer roles*, responsible for building, maintaining and serving the domain's data products. Data product developers will be working alongside their fellow application developers in the domain. Each domain team may serve one or multiple data products. It's also possible to form new teams to serve data products that don't naturally fit into an existing operational domain.

Data as a product creates a new system of the world, where data is and can be trusted, built and served with deep empathy for its users and their needs, and its success is measured through the value delivered to the users and not its size.

This ambitious shift must be treated as an organizational transformation. I will cover the organizational transformation Part IV of this book. It also requires an underlying supporting platform. The next chapter looks into the platform shift to support Data Mesh.

CHAPTER 6

Principle of Self-Serve Data Platform

A note for Early Release readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

> "Simplicity is about subtracting the obvious and adding the meaningful."
>
> —John Maeda, Technologist and Designer

So far I have challenged some of the characteristics of existing data architectures, and offered two fundamental shifts toward Data Mesh: A distributed data architecture and ownership based on business domains and closely integrated with the operational plane, and *data product*, a new architectural unit that encapsulates and shares *alive data* — code, data and policy — and embodies the characteristics of a usable and valuable product. Over time, these two seemingly simple and rather intuitive shifts can have undesired consequences: *duplication of efforts in each domain, increased cost of operation,* and most importantly *large scale inconsistencies and incompatibilities* across domains.

Expecting domain engineering teams to own and share analytical data as a product, in addition to building and managing applications and products, raises legitimate concerns for both the practitioners and their leaders. The concerns that I often hear

from leaders, at this point in the conversation, include: "How am I going to manage the cost of operating the domain data products, if every domain needs to build and own its own data?", "How do I hire the data engineers , who are already hard to find, to staff in every domain?", "This seems like a lot of over engineering and duplicate effort in each team?", "What technology do I buy to provide all the data product usability characteristics?", "How do I enforce governance in a distributed fashion to avoid chaos?", "What about copied data - how do I manage that?", and so on. Similarly domain engineering teams and practitioners' voice concerns such as "How can we extend the responsibility of our team to not only build applications to run the business, but also share data?".

Addressing these questions is the reason that Data Mesh's fourth principle, *self-serve data infrastructure as a platform,* exists. It is not that we have any shortage of data and analytics platforms and technologies, but we need to make changes to them so that they can *scale out* sharing, accessing and using analytical data, in a *decentralized manner*, for a new population of *generalist technologists*. This is the key differentiation of data platforms that enable a Data Mesh implementation.

Figure 6-1 depicts the extraction of domain-agnostic capabilities out of each domain, and abstracted by a self-serve infrastructure as a platform; that is built and maintained by a dedicated platform team.

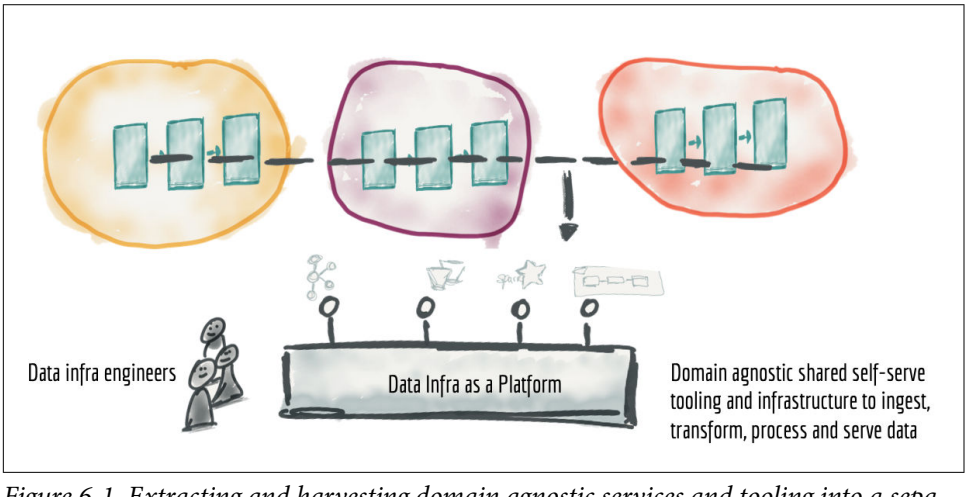

Figure 6-1. Extracting and harvesting domain agnostic services and tooling into a separate data platform

In this chapter, I apply platform thinking to the underlying infrastructure capabilities to clarify what we mean by the phrase 'platform' in the context of Data Mesh. Then, I share the unique characteristics of Data Mesh's underlying platform. Future chapters such as chapter 8, The Logical Architecture and chapter 10, The Multiplane Data Platform Design, will go into further details of the platform capabilities and how to

approach their design. For now, let's discuss how Data Mesh's underlying platform is any different from many solutions we have today.

In this chapter, I use the phrase *Data Mesh Platform* as a shorthand for a set of underlying data infrastructure capabilities that enable Data Mesh implementations. A singular form of the phrase *platform* does not mean a single solution, or a single- vendor and tightly integrated features. It is merely a placeholder for a set of technologies that one can use to achieve the objectives mentioned in section Data Mesh Platform Thinking.

Data Mesh Platform, Compare and Contrast

There is a large body of technology solutions that fall into the category of data infrastructure and often posed as a platform. These technologies include analytical data storage in the form of a lake, warehouse, or lakehouse; Frameworks and computation engines to process data in batch and streaming modes; data processing and querying languages based on two modes of computational data flow programming or algebraic SQL-like statements. There is a wide spectrum of data catalogue solutions to enable data governance as well as discovery of all data across an organization, and there are glue solutions that try to orchestrate complex data pipeline tasks or ML model deployment workflows. This list is just a small sample of the existing platform capabilities.

Many of these capabilities remain to be needed to enable a Data Mesh implementation. However, there is a shift in approach and the objectives of a Data Mesh platform. Let's do a quick compare and contrast.

Below is a set of unique characteristics of a Data Mesh platform in comparison to the existing ones. Note that Data Mesh platform can utilize the existing technologies, and yet offer these unique characteristics.

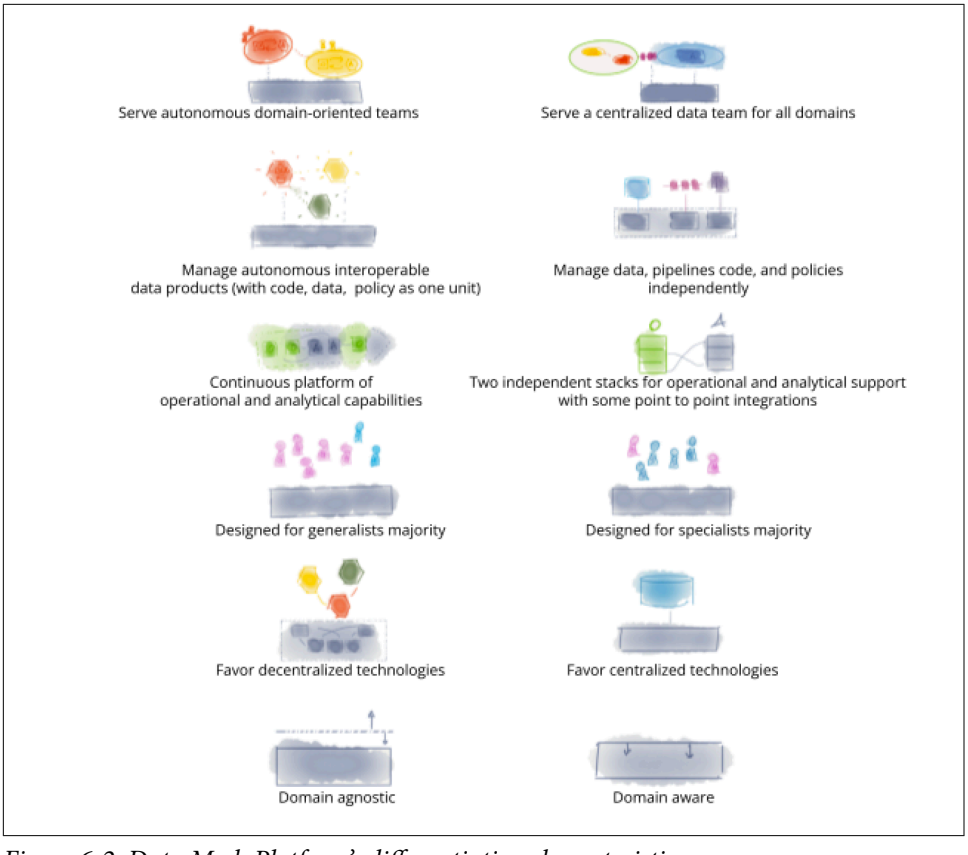

Figure 6-2. Data Mesh Platform's differentiating characteristics

Serving Autonomous Domain-oriented Teams

The main responsibility of the Data Mesh platform is to enable existing or new domain engineering teams with new and embedded responsibilities of building, sharing and using data products end to end; from capturing data from the operational systems and other sources, to transforming and sharing the data as a product with the end data consumers.

The platform must allow teams to do this in an autonomous way without dependencies to centralized data teams or intermediaries.

Many existing platforms and technologies are built with an assumption of a centralized data team, capturing and sharing data for all domains. The assumptions around the centralized control have deep technical consequences such as how the computation/storage are allocated, how accounts for the resources are structures, and so on. These assumptions get in the way of domain teams acting autonomously.

Managing Autonomous and Interoperable Data Products

Data Mesh puts a new construct, a domain-oriented data product, at the center of its approach. This is a new architectural construct that *autonomously* delivers value. It encodes all the functionality needed to provide discoverable, usable, trustworthy and secure data to its end data consumers. Data products share data with each other and are interconnected in a mesh. Data Mesh platforms must work with this new construct, must support managing its autonomous lifecycle, and all its constituents.

This platform characteristic differentiates from the existing platforms today that manage function, e.g. data processing pipelines, data and its metadata and policy that governs the data as independent pieces. Having said, it is possible to create the new data product management abstraction on top of the existing technologies.

A Continuous Platform of Operational and Analytical Capabilities

As discussed earlier, the principle of domain ownership demands a platform that enables autonomous domain teams to manage data end to end. This closes the gap both architecturally and organizationally between the operational plane and the analytical plane. Hence a Data Mesh platform must be able to provide a more connected experience whether the team is building and running an application or sharing and using data products for analytical use cases. For the platform to be successfully adopted with the existing domain technology teams, it must remove the barriers to adoption, the disconnection of operational and analytical stacks and their dichotomy.

The Data Mesh platform must close the gap between analytical and operational technologies. It must find ways to get them work seamlessly together, in a way that is natural to a cross-functional domain-oriented data and application team.

Designed for Generalists Majority

Another barrier to the adoption of data platforms today is the level of proprietary specialization that each technology vendor assumes - the jargon and the vendor-specific knowledge. This has led to creation of scarce specialized roles such as data engineers.

In my opinion there are two main reasons for this unscalable specialization: *lack of (defacto) standards and conventions*, and *lack of incentives for technology interoperability*. I believe this is the residue of the big monolithic platform mentality that a single vendor can provide soup to nuts functionality to store your data on their platform, and attach their additional services to keep the data there and its processing under the control of the vendor.

A Data Mesh platform must break this pattern, and start with the definition of a set of open conventions that not only prompt interoperability between different technolo-

gies but also reduce the number of proprietary languages and experiences one specialist must learn to generate value from data. Incentivizing and enabling generalist developers with experiences, languages and APIs that are easy to learn, and can be used independent of the underlying vendor are a starting point to lower the cognitive load of generalist developers. To scale out data-driven development to the larger population of practitioners, Data Mesh platforms must stay relevant to generalist technologists. They must get out of the way, move to the background and fit naturally into the native tools and programming languages generalists use. Needless to say that this should be achieved without compromising on the software engineering practices that result in sustainable solutions; contrary to what many low-code no-code platforms promise.

Favoring Decentralized Technologies

Another common characteristic of existing platforms is centralization of control. The examples include the centralized pipeline orchestration tools, centralized catalogues, centralized warehouse schema, centralized allocation of compute/storage resources, and so on. The reason for Data Mesh's focus on decentralization through domain ownership is to avoid organizational synchronization and bottlenecks that ultimately slow down the speed of change. Though on the surface this is an organizational concern, the underlying technology and architecture directly influences the organizational communication and design. A monolithic or centralized technology solution leads to centralized points of control, and teams. After all, decades later, Conway's law is well and alive.

> "Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."
>
> —Melvin Conway ("How Do Committees Invent?" HBR paper)

Data Mesh platforms must consider the decentralization of organizations in data sharing, control and governance at the heart of their design. They must inspect every centralized aspect of their design that can result in lockstep team synchronization, centralization of control, and tight coupling between autonomous teams and autonomous data products they create. Having said that, there might be many aspects of infrastructure that need to be centrally managed to reduce the unnecessary tasks that each domain team performs in sharing and using data, e.g. setting up data processing compute clusters. This is where an effective self-serve platform shines, centrally managing underlying resources while allowing independence of teams' to achieve their outcomes end to end, autonomously.

Domain Agnostic

Data mesh creates a clear delineation of responsibility between domain teams — who focus on creating business-domain-oriented products, services which are ideally

data-driven, and data products — and the platform teams who focus on technical enablers for the domains. This is different from the existing delineation of responsibility where the data team is often both responsible for amalgamation of domain-specific data for analytical usage, as well as the underlying technical enablers.

This delineation of responsibility needs to be reflected in the platform capabilities. The platform must strike a balance between providing domain-agnostic capabilities, while enabling domain-specific data modeling, processing and sharing across the organization. This demands a deep understanding of the data developers and consumers and application of product thinking to the platform.

Hopefully this short list clarifies the posture of Data Mesh toward building self-serve platforms. Let's go a bit deeper into top-level objectives of Data Mesh adopting the concept of platform thinking.

Data Mesh Platform Thinking

> "Platform: raised level surface on which people or things can stand."
>
> —Oxford Languages

The word *platform* is one of the most commonly used phrases in our everyday technical jargon and is sprinkled all over organizations' technical strategies. It's commonly used, yet hard to define and subject to interpretation.

To ground our understanding of the *platform* in the context of Data Mesh I draw from the work of a few of my trustworthy sources:

> "A digital platform is a foundation of self-service APIs, tools, services, knowledge and support which are arranged as a compelling internal product. Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced coordination."[1]
>
> —Evan Bottcher (What I Talk About When I Talk About Platforms)

> "The purpose of a platform team is to enable stream-aligned teams to deliver work with substantial autonomy. The stream-aligned team maintains full ownership of building, running, and fixing their application in production. The platform team provides internal services to reduce the cognitive load that would be required from stream-aligned teams to develop these underlying services.
>
> The platform simplifies otherwise complex technology and reduces cognitive load for teams that use it."[2]
>
> —Skelton, Matthew; Pais, Manuel. Team Topologies

1 https://martinfowler.com/articles/talk-about-platforms.html

2 Skelton, Matthew, et al. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. It Revolution, 2019.

> "Platforms are designed one interaction at a time. Thus, the design of every platform should start with the design of the core interaction that it enables between producers and consumers. The core interaction is the single most important form of activity that takes place on a platform—the exchange of value that attracts most users to the platform in the first place."[3]
>
> —Parker, Geoffrey G.; Van Alstyne, Marshall W.; Choudary, Sangeet Paul. Platform Revolution (Platform Revolution)

There are a few key objectives here, that I like to take away and apply to Data Mesh:

Enable Autonomous Teams to Get Value from Data
: A common characteristic that we see is the ability to enable teams, who use the platform, to complete their work and achieve their outcomes with a sense of autonomy and without requiring another team to get engaged directly in their workflow, e.g. through backlog dependencies. In the context of Data Mesh, enabling domain teams with new responsibilities of sharing analytical data, or using analytical data for building ML-based products, in an autonomous way, is a key objective of a Data Mesh platform. The ability to use the platform capabilities through self-serve APIs is critical to enable autonomy.

Exchange Value with Autonomous and Interoperable Data Products
: Another key aspect of a platform is to intentionally design *what value* is being exchanged and *how*. In the case of Data Mesh, data products are the unit of value exchange, between data consumers and data providers. A Data Mesh platform must build in the frictionless exchange of data products as a unit of value, in their design.

Accelerate Exchange of Value by Lowering the Cognitive Load
: In order to simplify and accelerate the work of domain teams in delivering value, platforms must hide technical and foundational complexity. This lowers the cognitive load of the domain teams to focus on what matters; in the case of Data Mesh, creating and sharing data products.

Scale out Data Sharing
: Data Mesh is a solution offered to solve the problem of organizational scale in getting value from their data. Hence the design of the platform must cater for the operational scale; sharing data across main domains within the organization, as well as across boundaries of trust outside of the organization in the wider network of partners. One of the blockers to this scale, is the lack of interoperability

3 Parker, Geoffrey G.; Van Alstyne, Marshall W.; Choudary, Sangeet Paul. *Platform Revolution: How Networked Markets Are Transforming the Economy and How to Make Them Work for You*. W. W. Norton & Company. Kindle Edition.

of data sharing, securely, across multiple platforms. A Data Mesh platform must design for interoperability with other platforms to share data products.

Support a Culture of Embedded Innovation
: A Data Mesh platform supports a culture of innovation by removing activities that are not directly contributing to the innovation; by making it really easy to find data, capture insights and use data for ML-model development.

Figure 6-3 depicts these objectives applied to the ecosystem of domain teams sharing and using data products.

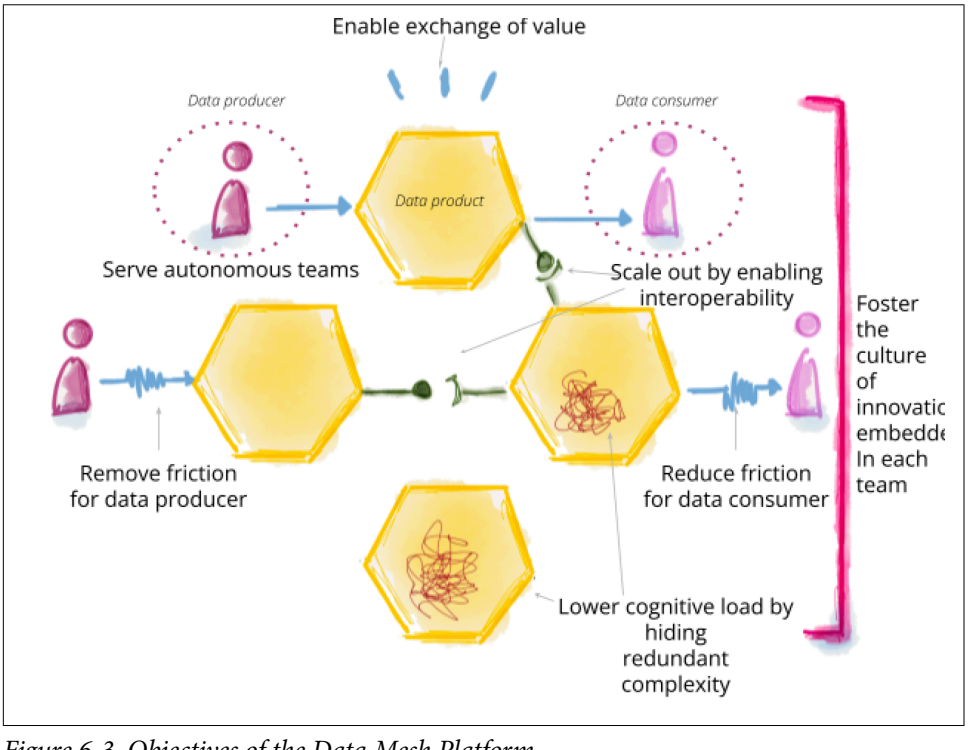

Figure 6-3. Objectives of the Data Mesh Platform

Now, let's talk about how a Data Mesh platform achieves these objectives.

Enable Autonomous Teams to Get Value from Data

In designing the platform to enable autonomous domain teams, it is helpful to consider the roles within each team, and their journey in sharing and consuming data products to ultimately create value. The platform can then focus on how to create a frictionless experience for each journey. For example, let's consider the two of the main personas of the Data Mesh ecosystem: *data product developers* and *data product*

consumers. Of course each of those personas include a spectrum of people different skill sets but for this conversion we can focus on the aspects of their journeys that are common across the spectrum. There are other roles such as data product owner whose journey is as important in achieving the outcome of creating successful data products, but favoring brevity, I leave them out of this example.

Enable Data Product Developers, Autonomously

The delivery journey of a data product developer involves creating a data product, building it, testing it, deploying, monitoring, updating and maintaining its integrity and security - with continuous delivery in mind. In short, *managing the life cycle* of a data product, its code, data and policies as one unit. As you can imagine there is a fair bit of infrastructure that needs to be provisioned to manage this lifecycle.

Provisioning and managing the underlying infrastructure for life cycle management of a data product requires specialized knowledge of today's tooling, and is difficult to replicate in each domain. Hence, the Data Mesh platform must implement all necessary capabilities allowing a data product developer to build, test, deploy, secure and maintain a data product without worrying about the underlying infrastructure resource provisioning. It must enable all domain-agnostic and cross-functional capabilities.

Ultimately, the platform must enable the data product developer to just focus on the domain-specific aspects of data product development. These include:

- Transformation code, the domain-specific logic that generates and maintains the data,
- Tests to verify and maintain the domain's data integrity,
- Tests to continuously monitor that the data product meets its quality guarantees,
- Generating data products meta-data such as its schema, documentation, etc.

The rest must be taken care of by the Data Mesh platform; for example, infrastructure provisioning — storage, accounts, compute, etc. A self-serve approach exposes a set of platform APIs for the data product developer to declare their infrastructure needs and let the platform take care of the rest. This is discussed in detail in section Managing Lifecycle of a data product in chapter 10.

Enable Data Product Consumers, Autonomously

Data consumers' journey — whether analyzing data to create insights or developing machine learning models — starts with discovering the data. Once the data is discovered, they need to get access to it, then understand it and deep dive to explore it further. If the data proven to be suitable, they will continue to use it. Using the data is not limited to a one-time access, the consumers continue receiving and processing

new data to keep their machine learning models or insights up to date. The Data Mesh platform builds the underlying mechanisms that facilitate such a journey and provide the capabilities needed for data product consumers to get their job done without friction.

For the platform to enable this journey, autonomously, it must reduce the need for manual intervention. For example, it must remove the need to chase the team who has created the data or the governance team to justify and get access to the data. Automating the process and exposing APIs that facilitate requests for access, and granting access based on automated evaluation of the consumer is an example.

Exchange Value with Autonomous and Interoperable Data Products

An interesting lens on the Data Mesh platform is to view it as a multi-sided platform — one that creates value primarily by enabling direct interactions between two (or more) distinct parties. In the case of Data Mesh, those parties are data product developers, data product owners and data product consumers.

This particular lens can be a source of unbounded creativity for building a platform whose success is measured directly by the exchange of value, i.e. data products. The value can be exchanged *on the mesh,* between data products, or *at the edge* of the mesh, between the end products such as an ML model, a report or a dashboard, and the data products. The mesh essentially becomes the *organizational data marketplace*. This particular Data Mesh platform characteristic can be a catalyst for a culture change within the organization, promoting sharing and consumption to the next level.

As discussed in the previous section, an important aspect of exchanging value is to be able to do that *autonomously*, without the platform getting in the way. For the data product developers this means being able to create and serve their data products without the constant need for hand holding or dependency on the platform team and consumers without needing hand holding by the producers.

Create Higher-order Value with Data Product Composability

The exchange of value goes beyond using a single data product, and often extends to composition and correlation of multiple data products. For example, the interesting insights about Daff Inc.'s listeners are generated by cross-correlating their behavior while listening to music, the artists they follow, their demographic, their interactions with social media, the influence of their fiends network, and the cultural events that surround them. There are multiple facets of the platform that make data product compatibility possible. For example, The role of the platform in enabling *data product linking* — when one data product uses data and data types (schema) from another data product. For this to be seamlessly possible, the platform must provide a standardized and simple way of *identifying* data products, *addressing* data products, *con-*

necting to data products, *reading* data from data products, etc. Such simple platform functions create a *mesh of heterogeneous domains with homogeneous interfaces*. I will cover the composability in chapter 9.

Accelerate Exchange of Value by Lowering the Cognitive Load

Cognitive load was first introduced in the field of cognitive science as the amount of working memory needed to hold temporary information to solve a problem or learn.[4] There are multiple factors influencing the cognitive load such as the intrinsic complexity of the topic at hand or how the task or information is presented.

Platforms are increasingly considered a way of reducing the cognitive load of developers to accelerate their tasks. They do this by hiding the amount of details and information presented to the developer: *abstracting complexity*.

For example, as a data product developer I should be able to express what my domain-agnostic wishes are, e.g. the structure of my data, its retention period, its potential size, without describing exactly how to implement those wishes. For example, I should be able to simply *declare* the confidentiality class of my data product and leave it to the platform to perform automatic encryption, managing encryption keys, automatic rotation of keys, etc. This is a domain-agnostic and cross-functional complexity that as a data developer or consumer I should not be exposed to.

There are many techniques for abstracting complexity without sacrificing configurability. The two methods below are commonly applied.

Abstract Complexity Through Declarative Modeling

Over the last few years, operational platforms such as container orchestrators - e.g. Kubernetes - or infrastructure provisioning tools - e.g. Terraform - have established a new model for *abstracting complexity* though *declarative modeling of the target state*. This is in contrast with other methods such as using *imperative* instructions to command how to build the target state. Essentially, the former focuses on the *what* and the later focuses on the *how*. In many scenarios declarative modeling hits limitations very quickly. For example defining the data transformation logic through declarations reaches a diminishing return as soon as the logic gets complex. However, systems that can be described through their state, such as provisioned infrastructure, lend themselves well to a declarative style. This is also true about Data Mesh infrastructure as a platform. The target state of infrastructure to manage the lifecycle of a data product can be defined declaratively.

4 Sweller, John (April 1988). "Cognitive Load During Problem Solving: Effects on Learning". Cognitive Science.

Abstract Complexity Through Automation

Removing the human intervention and manual steps from the data product developer journey through automation is another way to reduce complexity, particularly accidental complexity; complexity rose from manual errors through the process. Opportunities to automate aspects of a Data Mesh implementation are ubiquitous. The provisioning of the underlying data infrastructure itself can be automated using infrastructure as code[5] techniques. Additionally, many actions in the data value stream, from production to consumption, can be automated. For example, in the existing data pipelines data certification or verification approval process is often done manually. This is an area of immense opportunity for automation; automation of verifying the integrity of data, applying statistical methods in testing the nature of the data, even using machine learning to discover unexpected outliers. Such automation removes complexity from the data verification process.

Scale out Data Sharing

My observation of the existing big data technology landscape is the lack of standards for interoperable solutions that lead to data sharing at scale; for example, the lack of a unified model for authentication and authorization when accessing data, absence of standards for expressing and transmitting privacy rights with data, lack of standards in presenting temporality aspect of data, etc.

Most importantly, the data technology landscape is missing the Unix philosophy:

> "This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together …".
>
> —Doug McIlroy, then head of the Bell Labs Computing Sciences Research Center, and inventor of the Unix pipe

I think we got incredibly lucky with very special people, McIlroy, Ritchie, Ken and others seeding the culture, the philosophy and the way of building systems in the operational world. That's why we have managed to build powerfully scaled and complex systems through loose integration of simple small services and containers.

For some reason, we have abandoned this philosophy when it comes to big data systems, perhaps, because of those early assumptions that seeded the culture. Perhaps, because at some point we decided to separate data - the *body* - from its code - the *soul* - that led to establishing a different philosophy around it.

If a Data Mesh platform wants to realistically scale out sharing data, within and beyond the bounds of an organization, it must wholeheartedly embrace the unix philosophy, and yet adapt it to the unique needs of data management and data sharing. It

5 Morris, Kief. *Infrastructure as Code: Dynamic Systems for the Cloud Age*. O'Reilly Media, 2021.

must design the platform as a set of interoperable services that can be implemented by different vendors with different implementations, but yet play nicely with the rest of the platform services.

Take observability as an example of a capability that the platform provides; the ability to monitor the behavior of all data products on the mesh and detect any disruptions, errors, and undesirable access; and notify the relevant teams to recover operation. Even for a simple capability such as observibility, there are multiple platform services that need to cooperate, the data products emitting and logging information about their operation — ideally using a platform-provided functionality; the service that captures the emitted logs and metrics and provides a holistic mesh view; the services that search, analyze and detect anomalies and errors within those logs; and the services that notify the developers when things go wrong. To build this under the unix philosophy we need to be able to pick and choose these services, and connect them together. The key in simple integration of these services is *interoperability*[6], a common language and APIs by which the logs and metrics are expressed and shared. Without such a standard, we fall back to a single monolithic (but well integrated) solution that limits the scale of our operation to the environment where the monolithic solution is available. We fail to share data across environments while having access to the lineage and observability information.

Support a Culture Of Embedded Innovation

To date, continuous innovation must arguably be one of the core competencies of any business. Eric Ries introduced The Lean Startup[7] to demonstrate how to scientifically innovate through short and rapid cycles of *build-measure-learn.* The concept has since been applied to the larger enterprise through Lean Enterprise[8] - a scaled innovation methodology. The point is that to grow a culture of innovation — a culture of rapidly building, testing and refining ideas — we need an environment that frees its people from unnecessary work, accidental complexity and friction to experiment. Data Mesh platform removes unnecessary manual work, hides complexity and streamlines the workflows of data product developers and consumers, to free them to innovate using data. A simple litmus test to assess how effective a Data Mesh platform is in doing that, is to measure how long it takes for a team to dream up a data-driven experiment, and get to use the data to run the experiment. The shorter the time, the more mature the Data Mesh platform has become.

Another key point is the *team,* who is empowered to do the experiments. The Data Mesh platform supports a *domain team* to innovate and perform data-driven experi-

6 https://github.com/OpenLineage/OpenLineage is an attempt to standardize the tracing logs

7 http://www.startuplessonslearned.com/2008/09/lean-startup.html

8 Humble, Jez, et al. *Lean Enterprise*. O'Reilly Media, Inc., 2020.

ments. The data-driven innovations are no longer exclusive to the central data team. They must be *embedded* into each domain team in developing their services, products or processes.

Transition To Self-serve Data Mesh Platform

So far, I talked about the key differences between existing data platforms and Data Mesh, and covered the main objectives of the platform for a Data Mesh implementation. Here, I like to leave you with a few actions you can take in transitioning to your Data Mesh platform.

Design the APIs and Protocols First

When you begin your platform journey, whether you are buying, building or very likely both, start with selecting and designing the interfaces that platform exposes to its users. The interfaces might be programmatic APIs, shared as libraries or services. They might be command line or graphic interfaces. Either way, decide on interfaces first and then the implementation of those through various technologies.

This approach is well-adopted by many cloud offerings. For example cloud blob storage providers expose REST APIs[9] to post, get, or delete objects. You can apply this to all capabilities of your platform.

In addition to the APIs, decide on the communication protocols and standards that enable interoperability. Taking inspirations from internet — the one example of a massively distributed architecture — decide on the *narrow waist*[10] protocols. For example, decide on the protocols governing how data products express their semantic, in what format they encode their time-variant data, what query languages each support, what SLOs each guarantee, and so on.

Prepare for Generalists Adoption

I discussed earlier that a Data Mesh platform must be designed for the generalist majority. Many organizations today are struggling to find data specialists such as data engineers, while there is a large population of generalist developers who are eager to work with data. The fragmented, walled and highly specialized world of big data technologies have created an equally siloed fragment of hyper-specialized data technologists.

9 Example: https://docs.aws.amazon.com/AmazonS3/latest/API/s3-api.pdf#Type_API_Reference

10 The Evolution of Layered Protocol Stacks Leads to an Hourglass-Shaped Architecture, http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p206.pdf

In your evaluation of platform technologies, favor the ones that fit better with a natural style of programming known to many developers. For example, if you are choosing a pipeline orchestration tool pick the ones that lend themselves to simple programming of Python functions like Prefect — something familiar to a generalist developer — than then ones that try to create yet another domain-specific language (DSL) in Yaml or XML with esoteric notations.

In reality, there will be a spectrum of data products in terms of their complexity, and a spectrum of data product developers in terms of their level of specializations. The platform must satisfy this spectrum, to mobilize data product delivery at scale. In either case, the need for applying evergreen engineering practices to build resilient and maintainable data products remain.

Create Higher Level APIs to Manage Data Products

Data Mesh platform must introduce a new set of APIs to manage data products as a new abstraction. While many data platforms, such as the services you get from your cloud providers, include lower level utility APIs — storage, catalogue, compute — Data Mesh platform must introduce a higher level of APIs that deal with a data product as an object.

For example, APIs to create a data product, discover a data product, connect to a data product, read from a data product, secure a data product, and so on. See chapter 8, The Logical Architecture, for the logical blueprint of a data product.

Converge Data and Operational Platforms, Where Possible

The separation of analytical data plane and operational plane has left us with two disjointed technology stacks, one dealing with analytical data and the other for building and running our applications and services. As data products become integrated and embedded with the operational word, there is an opportunity and a need to integrate their respective platforms more closely. For example, today the computation fabric running data processing pipelines such as Spark are managed on a different clustering architecture, away and often disconnected from the computation fabric that runs operational services, such as Kubernetes. In order to create data products that collaborate closely with their corresponding microservice, i.e. source-aligned data products, we need a closer integration of the computation fabrics. I have worked with very few organizations that have been running both computation engines on the same Kubernetes fabric.

In my opinion the dual data and operational stacks have also led to investment in adopting tools that are marketed as *data* solutions while their operational counterparts are perfectly suitable to do the job. For example, I have seen a new class of CI/CD tooling marketed under DataOps. Evaluating these tools more closely, they hardly offer any differentiating capability that the existing CI/CD engines can't offer.

The dual stack investments have led to over investment in data tooling, further fragmentation of the platform technologies, and additional overhead of managing them.

I do hope that the Data Mesh platform is a catalyst in simplification of the technology landscape and closer collaboration between operational and analytical platforms.

Build Experiences, not Mechanisms

I have come across numerous platform building/buying experiences, where the articulation of the platform is anchored in *mechanisms* it includes, as opposed to *experiences* it enables. This approach in defining the platform often leads to bloated platform development, and adoption of overambitious and overpriced technologies.

Take 'data cataloging' as an example. Almost every platform I've come across has a 'data catalogue' on its list of mechanisms, which leads to the purchase of a data catalogue product with the longest list of features, and then overfitting the team's workflows to fit the cataloguing inner workings. This process often takes months.

In contrast, the platform can start with the articulation of the single experience of 'discovering data products'. Then, build or buy the simplest tools, mechanisms, that enable this experience. Then rinse, repeat and refactor for the next experience.

Begin with the Simplest Foundation, then Harvest to Evolve

Given the length of this chapter discussing the objectives and unique characteristics of a Data Mesh platform, you might be wondering "can I even begin to adopt Data Mesh today, or shall I wait some time to build the platform first?". The answer is to begin adopting a Data Mesh strategy today, even if you don't have a Data Mesh platform.

You can begin with the simplest possible foundation. Your smallest possible foundation framework is very likely composed of the data technologies that you have already adopted, especially if you are already operating analytics on the cloud. The bottom layer utilities that you can use as the foundation include the typical storage technologies, data processing frameworks, federated query engines, and so on.

As the number of data products grows, the standards are developed and common ways of approaching similar problems across data products are discovered. Then you will continue to evolve the platform as a harvested framework by collecting common capabilities across data products and domain teams.

Data Mesh platform itself is a product. It's an internal product — though built from many different tools and services from multiple vendors. The product users are the internal teams. It requires *technical product ownership*, long-term planning and long-

term maintenance. Though it continues to evolve and goes through evolutionary growth, its life begins today as a *minimum viable product (MVP)*[11].

Recap

Data Mesh's principle of *self-serve platform* comes to the rescue, to lower the cognitive load that the other two principles impose on the existing domain technology teams: own your analytical and share it as a product.

It shares common capabilities with the existing data platforms: providing access to polyglot storage, data processing engines, query engines, streaming, etc. However it differentiates from the existing platforms in its consumers: *autonomous domain teams made up of generalists technologist as majority*; it manages a higher level construct of a *data product* encapsulating data, code and policy as one unit.

Its purpose is to give domain teams super powers, by hiding low-level complexity behind simpler abstractions, and removing friction from their journeys in achieving their outcome of exchanging data products as a *unit of value*. And ultimately free up the teams to innovate with data. To scale out data sharing, beyond a single deployment environment or organizational unit or company, it favors decentralized solutions that are interoperable.

I will continue our deep dive into the platform in Chapter 10, The Multiplane Data Platform Design, and talk about specific services a Data Mesh platform could offer.

11 *The Lean Startup*

Prospective Table of Contents (Subject to Change)

Part I : Why Data Mesh?

Chapter 1: The Inflection Point

- Great Expectations of Data
- The Great Divide of Data
- Scale, Encounter of a New Kind
- Beyond Order
- Approaching the Plateau of Return

Chapter 2: After the Inflection Point

- Embrace Change in a Complex, Volatile, and Uncertain Business Environment
- Sustain Agility In the Face of Growth
- Increase the Ratio of Value From Data to Investment

Chapter 3: Before The Inflection Point

- Evolution of Analytical Data Architectures
- Characteristics of Analytical Data Architecture

Part II: What Is Data Mesh?

The Principles

The Origin

Chapter 4: Principle of Domain Ownership

- Apply DDD's Strategic Design to Data
- Domain Data Archetypes
- Transition to Domain Ownership

Chapter 5: Principle of Data as a Product

- Apply Product Thinking to Data
- Transition to Data Products

Chapter 6: Principle of Self-Serve Data Platform

- Data Mesh Platform, Compare and Contrast
- Data Mesh Platform Platform Thinking
- Transition to Self-Serve Data Mesh Platform

Chapter 7: Principle of Federated Computational Governance

- Apply Systems Thinking to Data Mesh Governance
- Apply Federation to the Governance Model
- Apply Computation to the Governance Model
- Transition to Federated Computational Governance

Part III: How to Design Data Mesh Architecture?

The Scope

The Approach

Chapter 8: The Logical Architecture

- Domain-Oriented Analytical Data Sharing Interfaces
- Data Product Quantum
- The Multi-plane Data Platform
- Embedded Computational Policies
- The Logical Architecture Boundary

Chapter 9: Data Quantum Design

- Data Quantum Affordances
- Serve Data
- Consume Data
- Transform Data
- Discover, Understand, Trust and Explore
- Compose Data
- Manage Lifecycle
- Govern Data
- Observe, Debug and Audit

Chapter 10: The Multi-Plane Data Platform

- To Come

Part IV: How to Get Started With Data Mesh

Chapter 11: Execution Model

- Data Mesh As An Org Level Strategy
- Cycle of Intelligence as a Vehicle to Execute
- Execution Model
- Maturity Model
- Milestones
- Migration From Legacy

Chapter 12: Organization Design

- Data Mesh Team Topologies
- Data Product Ownership
- Discovering Data Product Boundaries And Teams
- Population Profile

Chapter 13: What Comes Next

- The Gentle Move from Centralization to Decentralization

- What Is Next: Inter-org Data Mesh
- What Is After: Data Web
- Setting the Foundation For Sovereignty