Data Engineering Teams

Creating Successful Big Data Teams and Products

A practical guide to managing and creating Big Data teams

Jesse Anderson

Data Engineering Teams

Creating Successful Big Data Teams and Products

Jesse Anderson

Version 1.0.78ea02e

ISBN 978-1-7326496-0-6

Contents

CHAPTER 1

Introduction

About This Book

You're starting to hear, or have been hearing for a while, your engineers say "can't." The can'ts always revolve around processing vast quantities of data. Sometimes, you're asking for an analysis that spans years. You're asking for processing that spans billions of rows or looks at every customer you have. Other times, you're just asking for the day-to-day reports and data products that are the lifeblood of your business. Every time, there is a technical limitation that's keeping you from accomplishing your goal. You find yourself having to compromise every single time due to the technology.

You're sitting on a big pile of data. A **BIG** pile. Buried somewhere in there are a bunch of really, really valuable insights. The kind of insights that could fundamentally and deeply change your business. Maybe you could more deeply understand your customers and reveal every time they've purchased from you. Maybe it has to do with fraud or customer retention.

You, or your boss, have heard about *Big Data*, this magic bullet that everyone is talking about. It's allowing companies to free themselves of their technical limitations. It's also allowing your competition to move past you.

Here's the good news: I can help. In fact, I've been helping companies do this for the past 6+ years.

Here's the not-so-good news: the process isn't magic and Big Data only seems like a magic bullet to the outside observer. In fact, a lot of hard work goes into making a successful Big Data project. It takes a team with a lot of discipline, both technical and managerial.

Now back to the good news. I've seen, helped, and mentored a lot of companies as they've gone through this process. I know the things that work, and more importantly, I know the things that don't. I'm going to tell you the exact skills

that every team needs. If you follow the strategies and plans I outline in this book, you'll be way ahead of the 85% of your competitors who started down the Big Data road and failed.

When your competitors or other companies talk about Big Data technologies failing or their project failing due to the technologies, they're not recognizing a key and fundamental part of the project. They didn't follow the strategies and plans I outline in this book. They lay the blame squarely on the technology. The reality lies several months or a year earlier, when they started the project off on the wrong foot. I'll show you how to start it off right and increase your probability of success.

Warnings and Success Stories

This book is the result of years of carefully observing teams and entire companies. I've also spent years teaching at hundreds of companies and to thousands of students. These companies span the world and are in all kinds of industries. My work starts with my very first interaction with teams and continues as I follow up with those teams to see how their project went. From there, I analyze what went right, what went wrong, and why.

Throughout this book, I'm going to share some of these interactions and stories. They'll look something like this:

A cautionary story

Learn from this company's mistake(s)

These will be cautionary tales from the past. Learn from and avoid those mistakes.

Follow this example

Learn from this company's success

These will be success stories from the past. Learn from and imitate their success.

My story

Here's a story that will give you more color on the subject.

These will be stories or comments from my experience in the field. They will give you more background or color on a topic.

Who Should Read This

This book is primarily written for managers, VPs, and CxOs—people who are managing teams that are about to, or are currently, creating a Big Data solution. It will help them understand why some teams succeed but most teams fail.

Enterprise and Data Architects are tasked with being the bridge between the technical and business sides of the company. They have to make sure that the technology meets the needs of the business. They're the tip of the spear in advocating the need for Big Data technologies. This book will help them understand how to set the technical team up for success and the pitfalls to avoid.

Team leads, technical leads, and individual contributors have appreciated this book, too. They're often on the receiving end of why Big Data teams fail. This book will help these individuals understand the problems behind these failures so they can work with their management team to fix the issues.

Navigating the Book Chapters

I highly suggest you read the entire book from start to finish to understand every concept and point. Without the entire background and all of the concepts, you may not understand fully why I recommend a technique or make a specific suggestion.

Here are the chapters and what we'll be covering in each chapter:

- Chapter 2 will show you the reasons a data engineering team needs to exist. The key lies in the complexity of Big Data. A manager or team that doesn't internalize this fact is doomed to failure.
- Chapter 3 shows you what a data engineering team is and what makes it up. I'll talk about what makes a team successful and the common patterns of the unsuccessful teams.

- Chapter 4 will talk about the Data Engineers themselves. I'll show you how and why Data Engineers are different from Software Engineers.
- Chapter 5 shows you how to make your data engineering team productive after you've created it. I'll illustrate how data engineering teams function within an organization and how they interface with the rest of the company.
- Chapter 6 covers how to create data pipelines—that's what data engineering teams create. I'll share the ways I teach data engineers to create the most performant, reliable, and time-efficient pipelines.
- Chapter 7 shares the secrets to creating Big Data successful projects. I'll talk about how to break up the project into the manageable pieces, and share the reasons why teams succeed and fail.
- Chapter 8 shows you the exact steps to creating Big Data solutions. I've taught these steps to companies all over the world. They've used them to increase their probability of success. Don't just skip to this chapter and think you're going to be successful.

Conventions Used in This Book

A *DBA* (Database Administrator) is someone who maintains a database and often writes SQL queries. For the purposes of this book, I'm going to group several different titles that write SQL queries together for simplicity's sake. In addition to DBA, these titles would be Data Warehouse Engineer, SQL Developer, and ETL Developer.

I will use the terms *Software Engineer* and *programmer* interchangeably. Both are someone who has programming or software engineering skills. They are the team member responsible for writing the project's code.

With all that negative stuff and housekeeping out of the way, let's start mining for gold.

CHAPTER 2

The Need for Data Engineering

Big Data

Here is one buzzword subheading to get your attention. At its core, this is why we need data engineering teams. We have a proliferation of data and there's a massive of amount of potential business value in it. We have a high demand for data products that unlock business value, but a low supply of people who can create them.

Out of this demand comes the creation of teams dedicated to processing data. At least, they should be dedicated. It takes concerted effort and specialization to become a data engineering team.

Without this focus, we start a vicious cycle that's happening at companies and enterprises around the world. The thought is that Big Data is just like small data. The short answer is that it isn't. The companies that never internalize this fact are doomed to fail over and over. Worse yet, they're never going to understand why they failed.

Why Is Big Data So Much More Complicated?

Let's start off with a diagram that shows a sample architecture for a mobile product, with a database back end. Figure 2.1 illustrates a run-of-the-mill mobile application that uses a web service to store data in a database.

Let's contrast the simple architecture for a mobile app with Figure 2.2, which shows a starter Hadoop solution.

As you can see, Hadoop weighs in at double the number of boxes in our diagram. Why? What's the big deal? It's that a "simple" Hadoop solution actually isn't very simple at all. You might think of this more as a starting point or, to put it another way, as "crawling" with Hadoop. The "Hello World!" of a Hadoop solution is

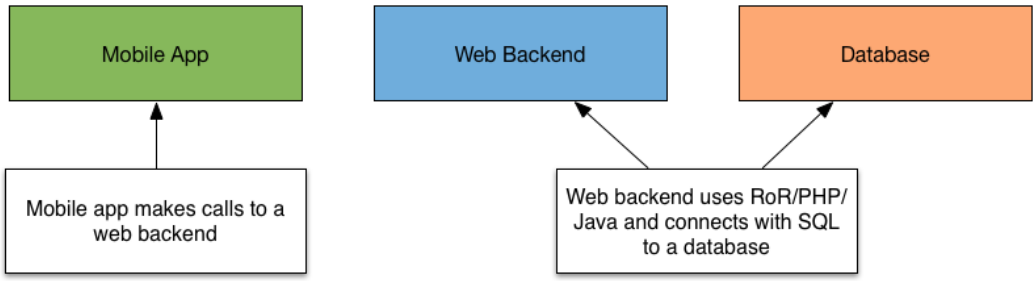

Figure 2.1: Simple architecture diagram for a mobile application, with a web backend.

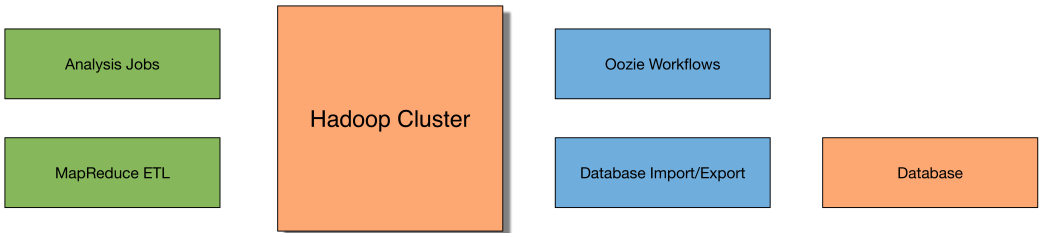

Figure 2.2: Simple architecture diagram for a Hadoop project.

more complicated than other domain's intermediate to advanced setups. Just look at the source code for a "Hello World!" in Big Data to see it's not so simple.

Now, let's look at a complicated mobile project's architecture diagram in Figure 2.3.

Note: That's the same diagram as the simple mobile architecture. A more complex mobile solution usually requires more code or more web service calls, but no additional technologies are added to the mix.

Let's contrast a simple big data/Hadoop solution with a complex big data/Hadoop solution in Figure 2.4.

Yes, that's a lot of boxes, representing various types of components you may need when dealing with a complex Big Data solution. This is what I call the "running" phase of a Big Data project.

You might think I'm exaggerating the number of technologies to make a point; I'm not. I've taught at companies where this is their basic architectural stack. I've also taught at companies with twice as much complexity as this.

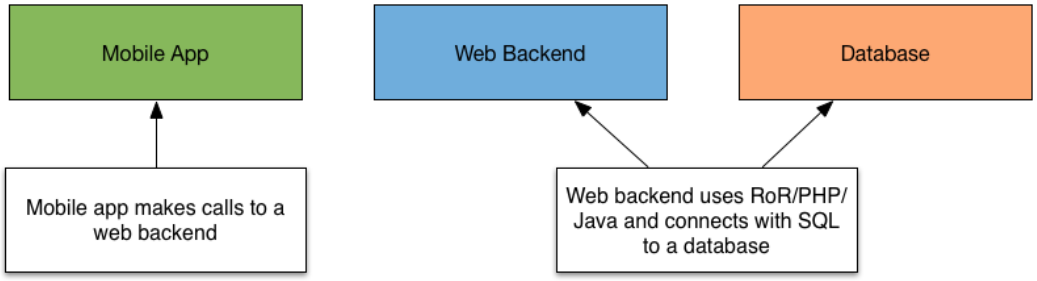

Figure 2.3: Complex architecture diagram for a mobile application with a web backend (yes, it's the same as the simple one).

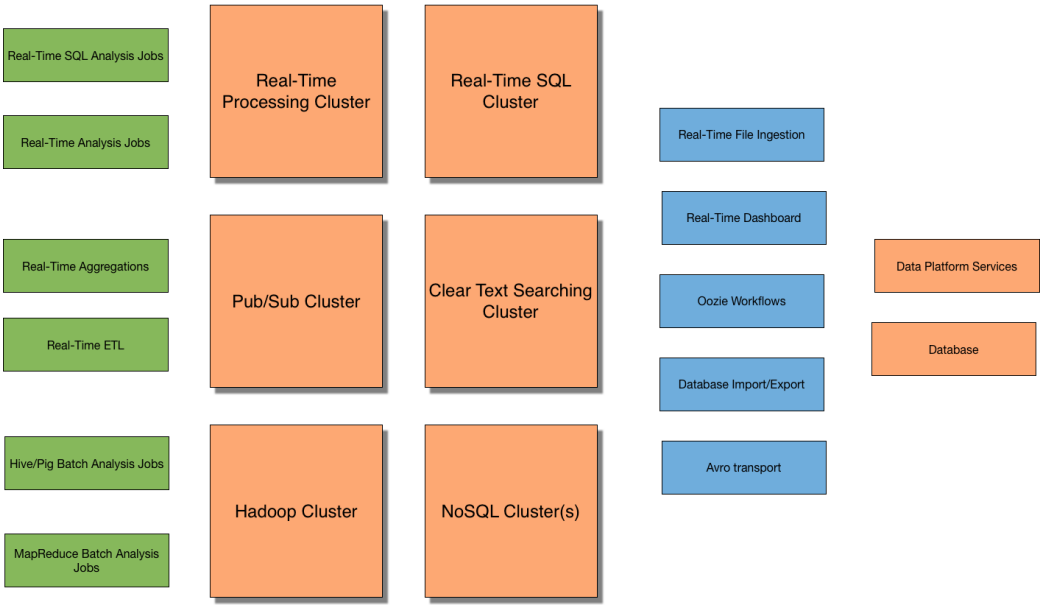

Figure 2.4: More complex architecture diagram for a Hadoop project with real-time components.

Instead of just looking at boxes, let's consider how many days of training it would take between a complex mobile and Big Data solution, assuming you already know Java and SQL. Based on my experience, a complex mobile course would take four to five days, compared to a complex Big Data course, which would take 18 to 20 days, and this estimate assumes you can grok the distributed systems side of all of this training. In my experience teaching courses, a Data Engineer can learn mobile, but a Mobile Engineer has a very difficult time learning data engineering.

Flavor of the month

While teaching at a company, I overheard one of the students say, "This is what we'll be doing this month." That company expected their developers to become experts during that week's training, develop the software, and then move onto another unrelated technology after that month was done. This is how you set your developers up for failure.

It struck me as odd that something like Big Data takes individuals six months to a year to learn and get good at. This team would only be given a month to be successful. I've seen unique individuals understand Big Data in two to three months. Your average enterprise developer, on the other hand, takes six months to a year to really understand Big Data and become productive at it.

You'll see me say that Data Engineers need to know 10 to 30 different technologies in order to choose the right tool for the job. Data engineering is hard because we're taking 10 complex systems, for example, and making them work together at a large scale. There are about 10 shown in Figure 2.4. To make the right decision in choosing, for example, a NoSQL cluster, you'll need to have learned the pros and cons of five to 10 different NoSQL technologies. From that list, you can narrow it down to two to three for a more in-depth look.

During this period, you might compare, for example, HBase and Cassandra. Is HBase the right one or is Cassandra? That comes down to you knowing what you're doing. Do you need ACID-ity? There are a plethora of questions you'd need to ask to choose one. Don't get me started on choosing a real-time

processing system, which requires knowledge of and comparison among Kafka, Spark, Flink, Storm, Heron, Flume, and the list goes on.

Distributed Systems Are Hard

Distributed systems frameworks like Hadoop and Spark make it easy, right? Well, yes and no.

Yes, distributed systems frameworks make it easy to focus on the task at hand. You're not thinking about how and where to spin up a task or threading. You're not worried about how to make a network connection, serialize some data, and then deserialize it again. Distributed systems frameworks allow you to focus on what you want to do rather than coordinating all of the pieces to do it.

No, distributed systems frameworks don't make everything easy. They make it even more important to know the weaknesses and strengths of the underlying system. They assume that you know and understand the unwritten rules and design decisions they made. One of those assumptions is that you already know distributed systems or can learn the fundamentals quickly.

I think Kafka is a great example of how, in making a system distributed, you add 10x the complexity. Think of your favorite messaging system, such as RabbitMQ. Now, imagine you added 10x the complexity to it. This complexity isn't added through some kind of over-engineering, or my PhD thesis would fit nicely here. It's simply the result of making it distributed across several different machines. Making a system distributed, fault tolerant, and scalable adds the 10x complexity.

What Does It All Mean, Basil?

We can make our APIs as simple as we want, but they're only masking the greater complexity of creating the system. As newer distributed systems frameworks come out, they're changing how data is processed; they're not fundamentally altering the complexity of these Big Data systems.

Distributed systems are hard and 10× more complex. They'll be largely similar to each other, but will have subtle differences that bite you hard enough to lose five months of development time if one of them makes a use case impossible. When you go to debug an issue or figure out why something isn't performing, you won't be looking at a single process running on a single computer; you'll be looking at hundreds of processes running on tens of computers.

What Does It Mean for Software Engineering Teams?

The members of a software engineering teams are smart people. Yet they often underestimate the level of complexity for a Big Data solution.

I invite you to sit back and trust me. Big Data is a very different and complex animal.

Underestimating Big Data Complexity

I've had Software Engineers say they can teach my Big Data technology classes. They're under the impression Big Data is easy. I've had Software Engineers say they'll be done with a four-day course in half a day. I've had DBAs tell me it's no different than a data warehouse. They think I don't know what I'm talking about or that I'm overstating the difficulty.

When this happens, I ask them to sit tight. You may be one of the very, very, very few outliers, because they do exist. But more than likely you're right there in the middle of the bell curve and this will be complex for your team.

What Does It Mean for Data Warehousing Teams?

A common logical conclusion is that your Big Data framework of choice should belong to your data warehouse group. After all, the Big Data framework is just an extension of your data warehouse. This sort thinking is one of the most common ways a Big Data project fails. The data warehouse team isn't ready for the massive jump in complexity.

I could go through the exercise of showing you diagram boxes for an enterprise data warehouse, but it would look similar to the web one. The data warehouse team deals with a single database technology or one box. More advanced teams add another box for ETL. The effect is the same; the data warehousing team is not ready for the rigors and complication of Big Data.

CHAPTER 3

Data Engineering Teams

If the data warehouse and software engineering teams aren't the right teams, what is the right team? The answer to that is a brand-new type of team. It's made up of members of other teams. The new team has a wider variety of specialties and skills. This team is called a *data engineering team*.

What Is a Data Engineering Team?

People often confuse data engineering teams and Data Engineers as being the same thing. They aren't. A data engineering team isn't made up of a single type of person or title. In fact, when a data engineering team isn't multidisciplinary, I deduct points from their probability of success.

This multidisciplinary approach is required because the team doesn't just handle data. They aren't programmers behind the scenes that are never interacted with. They interact with virtually every part of the company, and their products become its lifeblood. This isn't your average backend software; this will be how your company improves its bottom line and starts making money from its data.

That requires people who aren't just programmers. They're programmers who have cross-trained in other fields and skills. The team also has some members or skillsets that aren't normally in an orthodox software engineering team.

The team is almost entirely Big Data focused. This is where the team's specialty is consistent. Everyone on the data engineering team needs to understand and know how Big Data systems work and are created.

That isn't to say that a data engineering team can't create small data solutions. Some small data technologies will be part of a Big Data solution. Using the data engineering team for small data is entirely possible, but is generally a waste of their specialty.

Skills Needed in a Team

A data engineering team isn't a group of Software Engineers. Rather, it's a group of complementary skills and people. The team itself is usually dominated by Software Engineers, but has other titles that aren't usually part of a software engineering team.

The skills needed on a data engineering team are:

- Distributed systems
- Programming
- Analysis
- Visual communication
- Verbal communication
- Project veteran
- Schema
- Domain knowledge

Distributed Systems

Big Data is a subspecialty of distributed systems. Distributed systems are hard. You're taking many different computers and making them work together. This requires systems to be designed a different way. You actually have to design how data moves around those computers.

Having taught distributed systems for many years, I know this is something that takes people time to understand. It takes time and effort to get right.

Common titles with this skill: Software Architect, Software Engineer

Programming

This is the skill for someone who actually writes the code. They are tasked with taking the use case and writing the code that executes it on the Big Data framework.

The actual code for Big Data frameworks isn't difficult. Usually the biggest difficulty is keeping all of the different APIs straight; programmers will need to know 10 to 30 different technologies.

I also look to the programmers to give the team its engineering fundamentals. They're the ones expecting continuous integration, unit tests, and engineering

processes. Sometimes data engineering teams forget that they are still doing engineering and operate as if they've forgotten their fundamentals.

Common titles with this skill: Software Engineer

Can't everyone program?

A company had me teach their DBAs how to program. It was an uphill battle for the team and one they didn't win. Out of the 20 students, only one person was really capable of accomplishing her goals. Programming isn't the memorization of APIs; it's getting a computer to do what you want and creating a system. That company and team didn't understand that concept and thought they just needed to memorize some APIs. From there, they could just look everything up on StackOverflow. That just isn't the case.

Analysis

A data engineering team gives data analysis as a product. This analysis can range from simple counts and sums all the way to more complex products. The actual bar or skill level can vary dramatically on data engineering teams; it will depend entirely on the use case and organization. The quickest way to judge the skill level needed for the analysis is to look at the complexity of the data products. Are they equations that most programmers wouldn't understand or are they relatively straightforward?

Other times, a data product is a simple report that's given to another business unit. This could be done with SQL queries.

Very advanced analysis is often the purview of a Data Scientist. I'll talk about Data Scientists and data engineering teams in Chapter 5.

Common titles with this skill: Software Engineer, Data Analyst, Business Intelligence Analyst, DBA

Visual Communication

A data engineering team needs to communicate its data products visually. This is often the best way to show what's happening with data, especially vast amounts of it, so others can readily use it. You'll often have to show to data over time and with animation. This function combines programming and visualization.

A team member with visual communication skills will help you tell a graphic story with your data. They can show the data not just in a logical way, but with the right aesthetics too.

Common titles with this skill: Software Engineer, Business Intelligence Analyst, UX Engineer, UI Engineer, Graphic Artist

Verbal Communication

The data engineering team is the hub in the wheel where many spokes of the organization come in. We'll talk more about that concept later in Chapter 5, "Productive Data Engineering Teams," but the manifestation is important. You need people on the team who can communicate verbally with the other parts of your organization.

Your verbal communicator is responsible for helping other teams be successful in using the Big Data platform or data products. They'll also need to speak to these teams about what data is available. Other data engineering teams will operate like internal solutions consultants.

This skill can mean the difference between increasing internal usage of the cluster and the work going to waste.

Common titles with this skill: Software Architect, Software Engineer, Technical Manager

Project Veteran

A project veteran is someone who has worked with Big Data and has had their solution in production for a while. This person is ideally someone who has extensive experience in distributed systems or, at the very least, extensive multithreading experience. This person brings a great deal of experience to the team.

This is the person that holds the team back from bad ideas. They have the experience to know when something is technically feasible, but a bad idea in the real world. They will give the team some footing or long-term viewpoint on distributed systems. This translates into better design that saves the team money and time once things are in production.

Common titles with this skill: Senior Software Architect, Senior Software Engineer, Senior Technical Manager

Schema

The schema skill is another odd skill for a data engineering team, because it's often missing. Members with this skill help teams lay out data. They're responsible for creating the data definitions and designing its representation when it is stored, retrieved, transmitted, or received.

The importance of this skill really manifests as data pipelines mature. I tell my classes that this is the skill that makes or breaks you as your data pipelines become more complex. When you have 1 PB of data saved on Hadoop, you can't rewrite it any time a new field is added. This skill helps you look at the data you have and the data you need to define what your data looks like.

Often, teams will choose a small data format like JSON or XML. Having 1 PB of XML, for example, means that 25% to 50% of the information is just the overhead of tags. It also means that data has to be serialized and deserialized every time it needs to be used.

The schema skill goes beyond simple data modeling. Practitioners will understand the difference between saving data as a string and a binary integer. They also advocate for binary formats like Avro or Protobuf. They know to do this because data usage grows as other groups in a company hear about its existence and capabilities. A format like Avro will keep the data engineering team from having to type-check everyone's code for correctness.

Common titles with this skill: DBA, Software Architect, Software Engineer

Domain Knowledge

Some jobs and companies aren't technology focused; they're actually domain expertise focused. These jobs focus 80% of their effort on understanding and implementing the domain. They focus the other 20% on getting the technology

right. Domain-focused jobs are especially prevalent in finance, health care, consumer products, and other similar companies.

Domain knowledge needs to be part of the data engineering team. This person will need to know how the whole system works throughout the entire company. They'll need to deeply understand the domain for which you're creating data products; these data products will need to reflect this domain and be usable within it.

Common titles with this skill: DBA, Software Architect, Software Engineer, Technical Manager, The Graybeard, Project Manager

Skills Gap Analysis

Now that you've seen each skill, you need to do what's called a *gap analysis* (based on Paco Nathan's data science teams method). Take a piece of paper and turn it width-wise, or open a spreadsheet, and create eight columns. Then create rows for every person on the team. On the top row, write all of the skills a data engineering team needs that we just talked about. Next, list the names of everyone on the team in the far-left column. See Figure 3.1 for an example of how to do this.

	A	B	C	D	E	F	G	H	I
1		**Distributed systems**	**Programming**	**Analysis**	**Visual Comm.**	**Verbal Comm.**	**Project Veteran**	**Schema**	**Domain knowledge**
2	Mohamed								
3	Gabrielle								
4	Mateo								
5	Joseph								
6	Feng								
7	Adam								
8	Gabriel								
9	Jack								
10	Riya								
11	Juan								
12	**Total**								

Figure 3.1: Doing gap analysis using a spreadsheet

Now you're going to fill out the paper or spreadsheet software. Go through everyone on the team and put an "X" whenever that team member has that particular skill. This requires a very honest look—doing this analysis as honestly as possible could be the difference between success and failure for the team.

Now that you've placed the "X"s on everyone's skills, total them up. This will tell

you a story about the makeup of your team. It's going to tell you where you're strong, weak, or maybe even nonexistent.

Do Two Halves Make a Whole?

Sometimes people will do their gap analysis and put percentages or numbers with decimal points instead of an "X." They'll put a "0.1" or "0.5" instead of "X." Based on my experience, 10 people with a "0.1" don't make a "1" in total. They're really just several beginners with a "0.1" and all have the same limitations on the skill. This total is a very binary check. Either the person has those skills or doesn't, but several beginners don't add up to an expert.

Skill Gap Analysis Results

You've seen me make some bold statements about how data warehouse teams are not data engineering teams. This is where you're going to see why. If you have a team of DBAs, you will have several critical columns with nobody listed, or one person if you're lucky (Figure 3.2). More than likely those are the distributed systems and programming columns. Without these skills, data pipelines don't get created. This is how Big Data projects fail.

	A	B	C	D	E	F	G	H	I
1		**Distributed systems**	**Programming**	**Analysis**	**Visual Comm.**	**Verbal Comm.**	**Project Veteran**	**Schema**	**Domain knowledge**
2	Mohamed							X	
3	Gabrielle			X					X
4	Mateo			X					
5	Joseph							X	
6	Feng			X					
7	Adam			X					
8	Gabriel			X					
9	Jack			X					
10	Riya			X					
11	Juan			X		X			
12	**Total**	0	0	8	0	1	0	2	1

Figure 3.2: A gap analysis for an average DBA or data warehouse team with a simple analysis use case

Retail Woes

A very large retailer started their Big Data journey. They assigned the project to their data warehouse team. The team didn't have anyone with programming or distributed systems skills. They decided to overcome this shortcoming by purchasing an off-the-shelf Big Data solution that said it didn't require programming skills.

The off-the-shelf solution didn't solve their need for the distributed systems skill. The team wasn't able to understand how to create the system and how the data needed to flow through it in order for business value to be created.

The project eventually exhausted its budget and, worse yet, had little to nothing of value to show for the million dollars and months of work.

Let's take another example. Say we just took our software engineering team and didn't make it multidisciplinary. You may have several different low or absent skills—in this case, distributed systems, analysis, visual communication, verbal communication, project veteran, and schema (Figure 3.3). This is another way that Big Data projects fail; they just fail later on when you're creating an enterprise-ready solution.

	A	B	C	D	E	F	G	H	I
1		Distributed systems	Programming	Analysis	Visual Comm.	Verbal Comm.	Project Veteran	Schema	Domain knowledge
2	Mohamed	X	X						
3	Gabrielle		X						X
4	Mateo		X		X				
5	Joseph		X	X					
6	Feng		X	X					X
7	Adam		X						X
8	Gabriel		X			X			
9	Jack		X			X			
10	Riya	X	X				X		X
11	Juan		X						X
12	**Total**	2	10	2	1	2	1	0	5

Figure 3.3: A gap analysis for an average enterprise software engineering team

Other times you won't have any zeros. Congratulations! Or maybe not. I've found that teams overestimate their abilities and skills. Take another honest look and make sure you aren't deluding yourself. If you really have a gap, you

need to address it right away. Hiding from or lying to yourself won't make the problem go away.

You want to assess risk and any changes to the team that might be necessary. At this point, I ask the questions:

- What is your project or team mostly doing?
- Are you mostly creating data pipelines consumed by other teams?
- Are you mostly creating reports or analyzing data?
- How complex is your use case?
- Are you doing real-time or batch processing of data?
- Is the team serving as the hub of data and system information for the rest of the company?

Answering these questions gives you insight into where your team should be strongest. If your team is mostly analyzing data or creating reports, you'll need a few programmers, but mostly people with analysis skills. If you have a complex use case, you'll want more programmers and more veterans. If the team serves as the hub, you'll want more visual communication, verbal communication, schema, and domain knowledge practitioners.

Some of these skills are impossible or difficult to hire. For example, you'll have a hard time hiring someone with domain knowledge of your company's systems. In these situations, you'll prioritize knowledge transfer from your team members with domain knowledge skills to your veterans or other team members.

What I Look for in Data Engineering Teams

I want to share some of the things I've seen successful data engineering teams do and act like. These are some of the key elements that make a team successful.

I look for generally smart people who know how to work in a team. The list of skills needed for the team probably won't exist in a single person. A data engineering team is a multidisciplinary group of people. All of those skills are in different people and in different quantities within each person. The individuals will need to know how to work together to create a data pipeline.

Most teams don't need the level of interaction that a data engineering team needs. You need people on the team who can communicate verbally and visually.

As part of working together, each team member will need to have a very honest

view of themselves and the rest of the team. They need to know how their skills complement one another and are used in the team. This requires a great deal of honesty as the individual looks over their skills and the skills of other team members. Such an internal look tells them how to use others' complementary skills. The multidisciplinary nature of the team means that members will need to use one another's skills.

I look for teams where each member understands what the company and team is doing. You'd be surprised how often the entire team isn't thinking or doing the same thing. This comes from a general lack of understanding of what the team is doing.

When you're doing small data, these misunderstandings cost you a few hours or a day. When you're doing Big Data, these misunderstandings costs days, weeks, or months. The stakes and complexity is so much higher that there can be so much wasted time.

Diversity in Teams

Having taught extensively, I can say a diverse team performs much better than a uniform team. Monocultures leads to groupthink, which leads to a less innovative team, because everyone has similar ideas and thoughts.

The data engineering team is multidisciplinary and should be diverse. This diversity manifests in many different ways, from gender and ethnicity to socioeconomic backgrounds. This is how teams become more innovative and performant.

When I'm teaching, I've noticed that members of a uniform team often have the same misunderstanding or inability to understand a subject. They'll have the same or similar idea about execution. A diverse team will have some people who understand and others who don't. The people who do understand will help explain the subject to those who don't. A diverse team also will have more ideas about execution and they can choose the best one.

Diversity while teaching

I love teaching diverse teams and watching how they perform differently. I was teaching a difficult and advanced course to a diverse team that included students at different levels of understanding. During the course lecture, exercises, and even breaks, the students would help each other out and explain the concepts to the other members. I was happy to see this because I knew this behavior would continue long after I was gone.

A team that lacks diversity rarely has this ability. Normally, they have the same misunderstanding and can't help each other.

Operations

I'm often asked if a data engineering team also should be engaged in operations.

I'll start out with an indirect answer. With the cloud, we should be asking ourselves why we're still in the operations business at all. I think managed services pay off and we shouldn't be in the operations game.

That said, the cloud isn't a possibility for everyone. You have to do operations on an on-premises cluster.

I've seen all different models, but I haven't seen one work better overall than another. There are:

- Data engineering in one team and operations on a different team
- DevOps models where one team does both the engineering and operations.
- Data engineering and final tier support in one team and lower tier operational support in another team

You definitely don't want a throw-over-the-fence model. This where the Data Engineering team is so isolated from production issues that they don't know how their code is acting—or acting up—in production.

Some Big Data technologies aren't built with operations in mind. Sometimes they are operationally fragile or require a massive amount of knowledge. Some

of these technologies require such a massive amount of knowledge of the technology and the code, that you need DevOps from the beginning. When I teach those classes, I give operational requirements and programming requirements equal weight. The team is encouraged to use the DevOps model.

Choosing an operations model

Some technologies actually require you to go with a DevOps model. One of those technologies is Apache HBase. When I wrote Cloudera's HBase class, I interviewed everyone I could talk to. One thing stood out: You need to have a DevOps model because that made the most sense. To write the code, you needed an understanding of the operations side. To support the operations, you needed a deep understanding of the code. I wrote the class expecting teams to run in DevOps, and taught them how it should be set up.

For other technologies, the answer is not as clear cut. My general thought there is that most developers lack the Linux, networking, and hardware skills. I've watched too many developers struggle to make basic BIOS changes.

Quality Assurance

Another common issue is to figure out how a quality assurance (QA) team works with a data engineering team.

In general, there is a lack of automated tools for doing QA on Big Data pipelines. This is a question I'm often asked when teaching QA folks—how do I test this thing? The answer is complicated. It involves manual scripting and writing unit tests.

That means that your QA team needs to be Quality Assurance Engineers who write this test code. Most QA teams lack the programming skills to create these scripts and tests. This is where the data engineering team will have to write many of these tests to allow the QA team to automate their testing.

My recommendation is that Data Engineers should be writing unit tests no matter what. Usually the sorts of tests and bindings the QA team needs are

different. This is where some time on the schedule needs to be allocated for the Data Engineers to work with the QA team.

CHAPTER 4

Data Engineers

What Is a Data Engineer?

A Data Engineer is someone who has specialized their skills in creating software solutions around data. Their skills are predominantly based around Hadoop, Spark, and the open source Big Data ecosystem projects.

They usually program in Java, Scala, or Python. They have an in-depth knowledge of creating data pipelines.

Other languages

I'm often asked about other languages. Let's say a team doesn't program in Java, Scala, or Python. What should they do? The answer is very team-specific and company-specific. Do you already have a large codebase in that language? How difficult is it to use the Java bindings for that language? How similar is that language to Java?

The data pipelines they create are usually reports, analytics, and dashboards. They create the data endpoints and APIs for others to access data. This could range from REST endpoints for accessing smaller amounts of data to helping other teams consume large amounts of data with a Big Data framework. More advanced examples of data pipelines are fraud analytics or predictive analytics.

The field of Big Data is constantly changing. A Data Engineer needs to have in-depth knowledge of at least 10 technologies and know at least 20 more. You might have read that and thought I was exaggerating or you didn't believe me. Based on my interactions with companies, that is exactly what's needed.

I'm often asked how it's possible for a general-purpose Software Engineer to know so many different technologies. It isn't. It's only possible for generalist

Software Engineers to know all of these technologies once they start specializing. However, you can expect this of a Software Engineer who has specialized in Big Data technologies.

What I Look for in Data Engineers

Once again, I want to share with you some of the traits I've found in especially good Data Engineers. Not every Data Engineer will possess every one of these traits, but good ones will have several.

I can't stress enough how important it is for a Data Engineer to have a strong programming background. Data Engineers are commonly more midway to senior in their careers. Those fresh out of school usually have a Master's degree or above in Computer Science with focus on distributed systems or data. I have seen some especially bright junior engineers make great contributions to the team.

This will sound odd given how much I talked about the importance of programming, but the best Data Engineers are bored with just programming. That means that they've mastered or nearly mastered programming as a discipline. Writing another enterprise system or small data project doesn't hold much interest.

As a result, they've started to cross-train into other fields. These could be related to programming like data science or unrelated like marketing or analysis.

Data Engineers are bored with creating small data systems; they aren't as big or complex as the systems they'd like to design. The main driver for this is their desire to create data products that can be used by everyone.

This desire comes out of their shared love of data. You might know a Software Engineer who loves coding or maybe even a language. They are happiest when coding. Data Engineers love coding and data—if there isn't a love for data, there's at least an interest in it. I've found this distinguishes the great Data Engineers from the good Data Engineers.

They use this data because they are inherently curious about what is happening and why. They're going to use their data to either prove or disprove their hypotheses.

I don't focus on what technologies a Data Engineer knows. I focus on their understanding of systems, particularly distributed systems. They obviously

need to know some Big Data technologies and APIs. However, learning APIs or another technology is much easier once they know the basic architectural and design patterns of Big Data systems. A Data Engineer who has shown they can learn some Big Data technologies is likely to have the ability to learn other technologies.

I see this all the time when I train a team that is already working with Big Data technologies. They catch on to the concepts quicker because there are similarities to their other Big Data technologies. The team learns more from the training because they're not starting from scratch.

Qualified Data Engineers

I often talk about qualified Data Engineers. This means that they have shown their abilities in at least one real-world Big Data deployment. Their code is running in production and they've learned from that experience. They also know 10 to 30 different technologies, as I've mentioned.

A qualified Data Engineer's value is to know the right tool for the job. They understand the subtle differences in use cases and between technologies, and can create data pipelines. They're the ones you rely on to make the difficult technology decisions.

Not Just Data Warehousing and DBAs

Sometimes companies will take their data warehousing team or DBAs and call them Data Engineers. People with data warehousing and DBAs skills do have a place on the data engineering team, but they aren't Data Engineers.

A Data Engineer needs extensive programming, distributed systems, and Big Data skills. Most data warehouse teams and DBAs lack the programming skills necessary. For those skillsets, everything has to be in SQL. If a job can't be done with SQL, it can't be done. SQL as a language is good, but it can't do everything.

When a team has the wrong kind of engineers, it will look like the team is becoming or has become stagnant. It will never be able to move up to more complicated use cases. I call this "getting stuck at crawl." The team lacks the skills to move beyond the crawl phase (discussed in Chapter 7, "Running Successful Data Engineering Projects") and into more complex phases.

Teaching a data warehouse team

I was teaching a data warehouse team who kept on asking very advanced questions. We were covering the basics but a particular student was asking me about how to do everything in real-time (aka very difficult and advanced). I'd give them the answer and each time I'd get a blank look. Having taught extensively, I know that a blank look means they didn't understand the answer. For this team, I couldn't make the answer any simpler because they were asking advanced questions.

I overheard the student asking advanced questions while talking to a coworker, and the coworker was asking the same questions the student had asked me. He replied that those use cases were impossible and implicated the technology as the reason why it couldn't be done. The technology was perfectly capable of doing it; the student just wouldn't admit that they personally couldn't do it or understand how to do it.

Ability Gap

As a trainer, I've lost count of the number of students and companies I've taught. One thing is common throughout my teaching: there is an ability gap. Some people simply won't understand Big Data concepts on their best day.

Most industry analysts usually talk about skills gaps when referring to a new technology. They believe it's a matter of an individual simply learning and eventually mastering that technology. I believe that too, except when it comes to Big Data.

Big Data isn't your average industry shift. Just like all of the shifts before it, it's revolutionary. Unlike the previous shifts, the level of complexity is vastly greater.

This manifests itself in the individuals and students trying to learn Big Data. I'll talk to people where I've figured out they have no chance of understanding the Big Data concept I'm discussing. They've simply hit their ability gap. They'll keep asking the same question over and over in a vain attempt to understand.

Big Data is hard. Not everyone will be able to be a Data Engineer. They may be

part of the data engineering team, but not be a Data Engineer.

Yes, this is harsh

I know when a person has an ability gap. It's a person struggling valiantly to understand Big Data, but who, in the end, won't ever be able to understand it well enough to create a system. They'll be able to understand the concepts, but not understand it deeply enough to create a data pipeline. I've tested this theory out for longer periods of time with students. No matter how much time or effort, they won't be able to get it. Becoming a Data Engineer isn't a good use of their time.

As a manager or team lead, you need to make these difficult judgment calls. I've found that the person themselves can't make the call and can't figure out why they're not getting anywhere in the project. If you're experiencing a project that has little to no progress, your team may be hitting this issue.

CHAPTER 5

Productive Data Engineering Teams

Themes and Thoughts of a Data Engineering Team

Data engineering teams have to change their thinking when they're dealing with Big Data. In my courses, my chapter on distributed systems is called "Thinking in Big Data." It's an odd name for a chapter because most teams don't realize that they need to change their thinking about data and systems to be successful with Big Data. They need to think about how data is valuable and what scale the data is coming in at.

When thinking about scale, I encourage teams to think in terms of:

- 100 billion rows or events
- Processing 1 PB of data
- Jobs take 10 hours to complete

I'll talk about each one of these thoughts in turn.

When you are processing some data, in real-time or batch, you need to imagine that you're processing 100 billion rows. These can be 100 billion rows of whatever you want them to be. This affects how you think about reduces or the equivalent of reduces in your technology of choice. If you reduce inefficiently or when you don't have to, you'll experience scaling issues.

When you are thinking about amounts of data, think in terms of 1 PB. Although you may have substantially less data stored, you'll want to make sure your processing is thought about in those terms. As you're writing a program to process 100 GB, you'll want to make sure that same code can scale to 1 PB. One common manifestation for this problem is to cache data in memory for an algorithm. While that may work at 100 GB, it probably will get an out of memory error at 1 PB.

When you are thinking about long-running processes, I encourage teams to think of them running for 10 hours to complete. This thought has quite a few

manifestations.

The biggest manifestation is about coding defensively. If you have an exception at 9.5 hours into a 10-hour job, you have two problems: to find and fix the error, and to rerun the 10-hour job. A job should be coded to check assumptions, whenever possible, to avoid an exception from exiting a job. A common example of these issues is when a team is dealing with string-based formats, like JSON and XML, and then expecting a certain format. This could be casting a string to a number. If you don't check that string beforehand with a regular expression, you could find yourself with an exception.

Given the 10-hour job considerations, the team needs to decide what to do about data that doesn't fit the expected input. Most frameworks won't handle data errors by themselves. This is something the team has to solve in code. Some common options are to skip the data and move on, log the error and the data, or to die immediately. Each of these decisions is very use case dependent. If losing data or not processing every single piece of data is the end of the world, you'll end up having to fix any bad data manually.

Every engineering decision needs to be made through these lenses. I teach this to every team, even if their data isn't at these levels. If they truly are experiencing Big Data problems, they will hit these levels eventually.

More importantly, the teams won't have to change their code as they start hitting these levels. Even using a Big Data framework, teams can still write code that doesn't scale or scale well.

Another important theme is that a qualified Data Engineer needs to know around 30 different technologies at varying proficiencies to be able to choose the right tool for the job. These range from batch frameworks and real-time frameworks to NoSQL technologies.

A final theme is that a subtle nuance in a technology can make a world of difference in the use case. That's the difference between a Data Engineer and a Software Engineer. A Data Engineer can see these subtle nuances and what they mean to a system much faster. A Software Engineer can't.

Hub of the Wheel

Sometimes I'm teaching at large enterprises and they disagree that there should be a separate data engineering team. They're usually thinking entirely of the

technical requirements. Setting a data engineering team at the hub of the wheel puts a data engineering team in a completely different light. It shows how a data engineering team becomes an essential part of the business process.

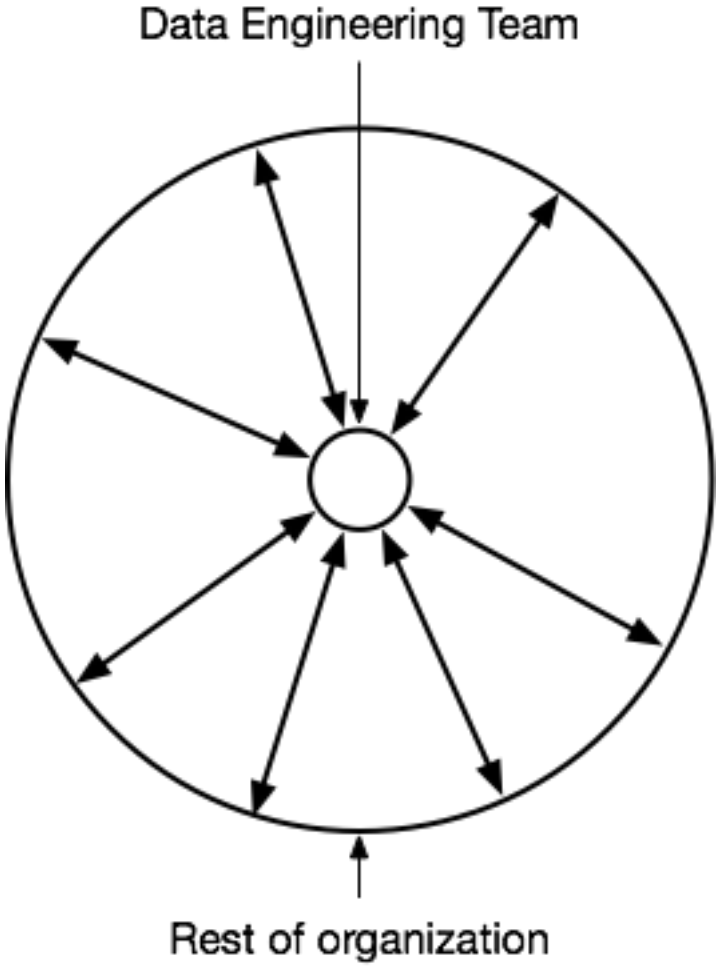

Figure 5.1: The data engineering team as the hub of data pipeline information for the organization

This hub concept is often foreign or not thought about. The organization doesn't realize that their project will grow in scope. As the other parts of the organization begin to consume the data or use the data pipeline, they're going to need help. This help comes as a result of an ability gap, a team with no programming skills, or another programming team that needs help.

Once a data pipeline is first released, it doesn't stay at its initial usage; it almost always grows. There is pent-up demand for data products that the pipeline starts to facilitate. New datasets and data sources will get added. There will be new processing and consumption of data. In a complete technical free-for-all, you will end up with issues. Often teams that lack qualified Data Engineers will completely misuse or misunderstand how the technologies should be used.

Being the hub in the wheel means that the data engineering team needs to understand the whole data pipeline. They need to disseminate this information to other teams. They need to help other teams know what data is available and what the data format is. Finally, they'll have to review code or write the code for other teams.

Organizations that fail to establish a data engineering team create an inefficient use of data or hinder using data for decision making. At these organizations, there is a huge demand for data and data products, but a complete lack of or low-performing data products for the business side of the company to use. As a direct result, the organization never fully utilizes its data for decision making.

Other times companies balk that their Software Developers need to specialize as Data Engineers. They don't understand how much their qualified Data Engineers will need to help other teams, or how much time that will take.

How to Work with a Data Science Team

Before I talk about their relationship to data engineering teams, I want to briefly define a Data Scientist and data science team.

A Data Scientist is someone with a math and probability background who also knows how to program. They often know Big Data technologies in order to run their algorithms at scale.

A data science team is multidisciplinary, just like a data engineering team. The team has the variety of skills needed to prevent any gaps. It's unusual to have a single person with all of these skills and you'll usually need several different people.

A Data Engineer is different from a Data Scientist in that a Data Engineer is a much better programmer and distributed systems expert than a Data Scientist. A Data Scientist is much better at the math, analysis, and probabilities than a Data Engineer. That isn't to say there isn't some crossover, but my experience

is that Data Scientists usually lack the hardcore engineering skills to create Big Data solutions. Conversely, Data Engineers lack the math backgrounds to do advanced analysis. The teams are more complementary than heavily overlapping.

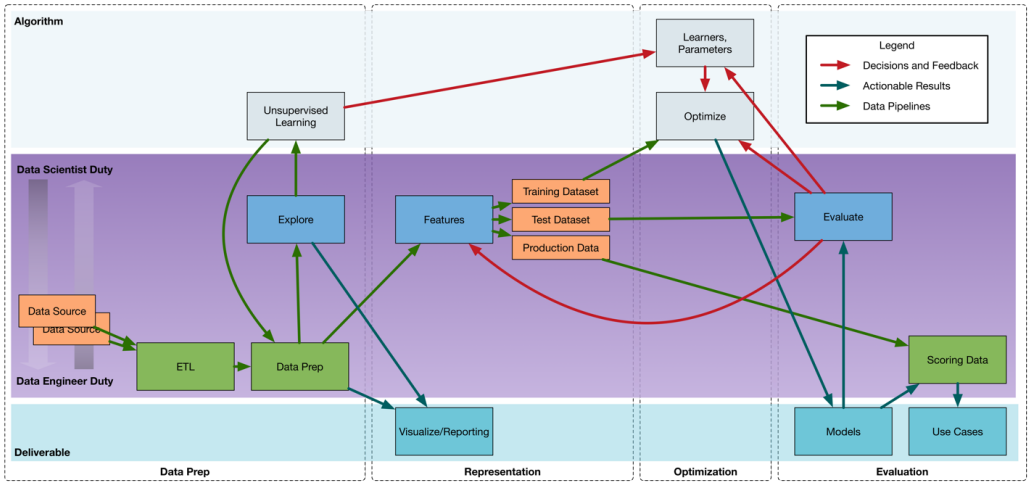

Figure 5.2: How Data Scientists and Data Engineers work together. Based on Paco Nathan's original diagram. Used with permission.

In Figure 5.2, I show how tasks are distributed between data science and data engineering teams. I could, and have, talked about this diagram for an hour. The duties are shown as more of a data science or data engineering duty by how close they are to the top or bottom of the center panel. Notice that very few duties are solely a data science or data engineering duty.

There are a few points I want you to take away from this diagram. A data engineering team isn't just there to write the code. They'll need to be able to analyze data, too.

Likewise, Data Scientists aren't just there to just make equations and throw them over the fence to the data engineering team. Data Scientists need to have some level of programming. If this becomes the perception or reality, there can be a great deal of animosity between the teams. In Figure 5.3, I show how there should be a high bandwidth and significant level of interaction between the two teams.

Ideally, there's a great deal of admiration and recognition between the two teams.

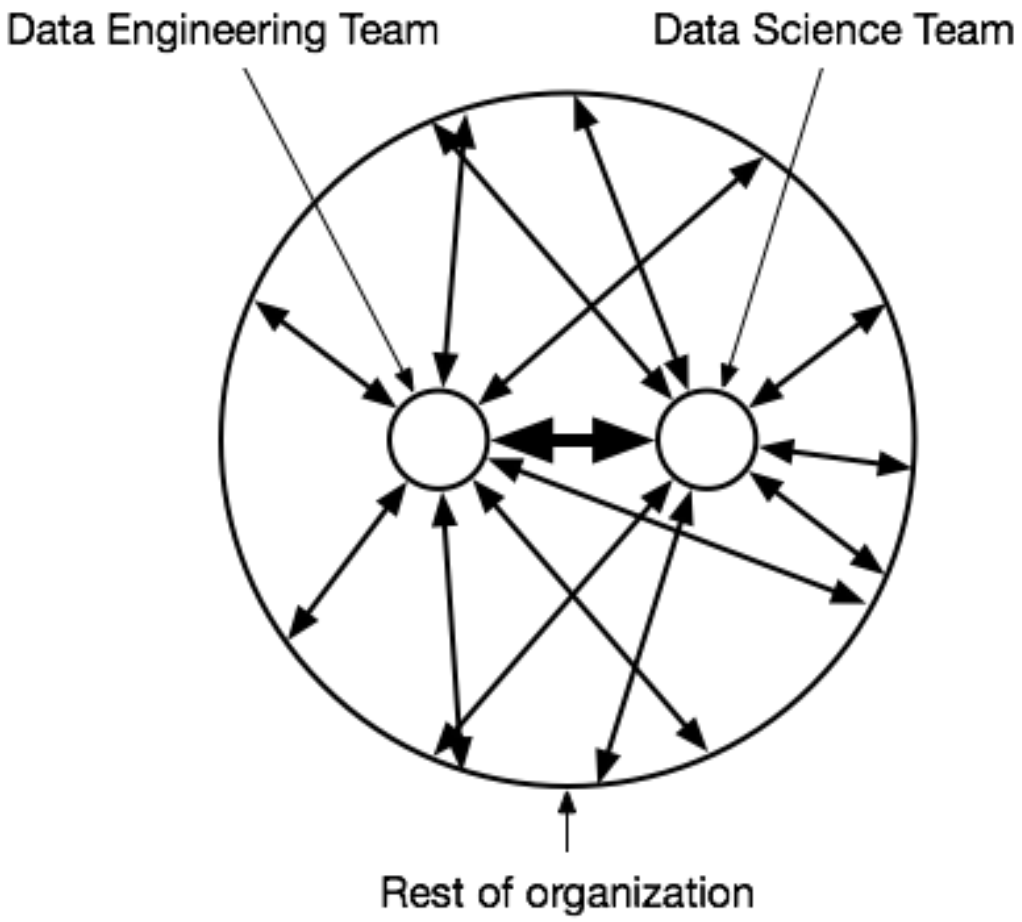

Figure 5.3: How a data engineering team and data science team should interact with each other and the rest of the company

Each realizes the other's abilities. It's magic on both sides. Data Scientists have mathematical magic. Data Engineers have programming magic. Neither side will completely understand what the other did. That's completely understandable given each team's skills and abilities.

Data science teams have a tradeoff to make. They need to be agile in their analysis. Data Scientists tend not use software engineering practices because they believe it will weigh them down. Data Engineers come from hardcore engineering disciplines and use software engineering best practices. The two teams will need to strike a balance between agility and sticking to good engineering practices. The majority of Data Scientists haven't coded on a hardcore engineering team before and will need help understanding how and why things are done. The Data Scientists will have to decide which times make sense to put under engineering processes. Both teams need to learn how to compromise.

Sometimes data science and data engineering teams are together in the same team. This usually depends on the size of the teams. The biggest factor in keeping the two teams productive is political. Are there rifts that prevent the two teams from working well together? Other than this, I've seen both teams work very well together.

Together or separate?

I find the usual reason for data engineering and data science being together or separate comes down to how they were started. Usually, the data science team comes out of the analytics or business intelligence side of the organization, and the data engineering team comes from its engineering side. For the two teams to be merged, they'd have to cross organizational boundaries, and that doesn't happen very often.

One suggestion that I don't see used often is to pair-program. I'm not saying everything should be done as pair programming, but I'm suggesting that teams do so when it makes sense. A Data Engineer and a Data Scientist would program together. This way, the resulting code is much better and the knowledge is transferred immediately. The code would be checked as it's written for quality by the Data Engineer and for correctness by the Data Scientist.

How to Work with a Data Warehousing Team

In contrast to the data science team, there is a great deal of overlap with a data warehousing team. Often, the entire reason for creating a data engineering team and moving to Big Data solutions is to move off of data warehousing products.

Big Data technologies can do everything that a data warehousing product can do and much more. However, the skillsets are very different. Whereas a data warehousing team focuses on SQL and doesn't program, a data engineering team focuses SQL, programming, and other necessary skills.

The data warehousing team is almost always separate from a data engineering team. Some companies will rename their data warehousing team as a data engineering team. As noted, the skillsets are very different, and the levels of complexity between the two teams much greater. Elements from a data warehousing team can sometimes fill in skills gaps in a data engineering team, however; usually domain knowledge and analysis skills.

If you do decide to merge or rename your data warehousing team as a data engineering team, you will need to check for ability gaps. This is a common reason why data warehousing teams have low success rates with Big Data projects.

How to Work with an Analytics and/or Business Intelligence Team

Usually a data engineering team is creating a data pipeline that, in whole or part, will help the analytics and/or business intelligence (BI) teams. These teams have hit some kind of performance issue or bottleneck in their small data systems. The data engineering team is creating the infrastructure to enable the analytics and/or BI teams to scale.

The analytics and/or BI teams usually lack the programming skills necessary to interface with the data pipeline. The data engineering team will take care to provide the programming support the other teams need.

The data engineering team should make sure that the analytics and/or BI teams can self-serve. They shouldn't have to get a member of the data engineering team to run a query for them. That's a waste of the data engineering team's time and doesn't engender a spirit of cooperation between the teams.

The data engineering team should be helpful, but the analytics and/or BI teams

should learn the technologies up to their abilities. Many Big Data technologies offer SQL interfaces that give non-programmers a way to access the data without knowing how to program. Even if the teams already know SQL, they should learn how to use the SQL interfaces for themselves.

How I Evaluate Teams

When I evaluate a data engineering team for effectiveness I consistently look for the same things. As I look over the team, I create a probability of success given what I know about their use case, the complexity, and the organization.

Does the team have the right people? If you have a key skills gap in the team, your probability of success plummets. In my experience, teams with a gap in the programming and distributed systems skills have a 5% to 25% chance of success. Other skills gaps aren't as big of a hit on the probability of success.

Unit Testing

I also check whether the team is unit testing their code. Data pipelines are really complex. How do you know if the code changes you made will cause other problems? Without good unit tests, the team's productivity will grind to a halt. The team will have a complex system with no guarantees it works. This is unfortunately one of the most common failures in data engineering teams. A large portion of the blame goes to the Big Data technologies which don't have unit testing as part of the project.

Let me give you an example of why this is important. I evaluate data engineering teams on how fast they can replicate a bug. If you have a problem in your production data pipelines and your Data Engineer can't replicate a bug in their IDE in 10 minutes or less you have two problems: a production problem and a unit testing problem. Data pipelines should be logging out problems or problem data. A Data Engineer should be able to take that log output, put it into an already written unit test, and replicate the problem. If your team can't do that, you're going to be forever chasing and fixing issues.

Another core reason for unit testing is to have a full stack or integration test. You'll recall that data pipelines are made up of 10 to 30 different technologies. Each one of those technologies has a different API for doing unit tests. If you don't have a unit test that flows all the way from the very beginning of the data pipeline

to its end, you won't be able to find and fix the most difficult problems. A data engineering team that has these capabilities won't just be initially successful; they'll continue to be successful as the data pipeline grows over time.

Automation

Automation is another multifaceted and key metric for a team. Going from small data to Big Data represents a significant increase in complexity. The current team is experiencing and creating that increase in complexity as it happens. Experiencing these changes in deltas, instead of all at once, makes it easier for the current team to understand the system.

The major difficulty arises when new people join the team. How long will it take for them to be productive? Let's say with your small data system it took one month. With Big Data, you could create a system that takes six months to understand—and that's apart from all the domain knowledge that's necessary. This thought should factor into your hiring and overall system design. Are you creating a system so difficult only its designers will understand it?

How long does it take for a new person on the team to set up their development environment? Setting up a Big Data environment on a developer's machine isn't an easy task. This can take several days of trial and error. There are many different processes that need to run, and the steps to configure aren't documented well. Additionally, this isn't a developer's core competency. This is something that should be scripted, or, in my opinion, done on a virtual machine (VM). This allows developers to have a VM already set up and running the necessary processes. More importantly, it makes every developer run the same versions of the software.

A related issue is how fast can you deploy a fix or new code to production. How many manual steps are there? With small data systems, teams can squeak by doing things manually. With Big Data, this changes because you're dealing with 30 different computers or instances. The same tools and techniques just break down. You'll need to be able to deploy code quickly—ideally with the push of a button.

A team that lacks production automation will suffer from having 30 machines all configured in slightly different ways. It will have program versions that are potentially different codebases. It will be soaking up operations time on deploying, instead of fixing the problem. All these are recipes for difficult production

problems.

Equipment and Resources

Equipping and giving sufficient resources to a data engineering team is the key to its productivity. A common misunderstanding is that Big Data is cheap. It isn't. Yes, your licensing fees will drop dramatically because the software is open source, but your costs elsewhere will go up.

A common issue is that data engineering teams need sufficient hardware to develop and run data pipelines on. This includes their development machines. I can't tell you how often I've taught at companies where they're expecting their data engineering team to use the same hardware they used for small data development on their Big Data development. The hardware needs to be able to run the software for the entire data pipeline at the same time. If it can't, you won't be able to debug a full data pipeline. Explain that to the CxO when there's an issue in production.

Another odd but consistent issue is that developers lack admin access to their own development machines. They will need to install new software.

The developers will need, and should be using, VMs to do their development. I teach developers in my classes how to use VMs efficiently to develop, try out new frameworks, and switch between different software versions. As your data pipeline becomes more complex, you'll need to use VMs to install new software or quickly switch between the production version and the newest version for development. These tricks are what make data engineering teams even more productive.

Teams also need clusters. Overall, the teams will need development, quality assurance, and load-testing clusters in addition to the production cluster.

The load-testing and production clusters should have the same specifications as what is in production. I've seen so many teams unable to rate the performance of their software because they don't have the same hardware as in production. What is the maximum number of messages, events, widgets, or whatever that the software can handle or process per unit of time? The only way to know is through load testing.

You don't throw out good engineering practices when you create your data engineering team. Now more than ever, you need to use your continuous integration

server to test your full data pipeline with unit tests. This is how data engineering teams gain velocity in development.

Last, not all resources are physical. A key ingredient to data engineering resources is learning materials. Big Data is constantly changing and your team will need to keep up with the changes. You need to give the team the training they require. If you're starting on a brand-new technology, getting in-person training is the cheapest way to test it out. Few things are more expensive than finding out halfway into a project that the technology isn't a good fit or that you're using it incorrectly. Books are another great resource, so make sure the team has access to the relevant texts.

CHAPTER 6

Creating Data Pipelines

Thought Frameworks

Thought frameworks are some decision-making techniques I teach data engineering teams. They help teams make decisions about data pipelines, write scalable code, and create good technical processes.

One framework concerns optimization in Big Data. Teams often optimize prematurely or on the wrong piece of the puzzle. Optimizations should focus on a little of a lot or a lot of little.

A little of a lot means that if a part of your processing takes 10 minutes as part of a 20-minute job, you have little of a lot. A small part of your job is taking up a good portion of your processing time. That's a great place to start optimizing.

A lot of a little means that you would call a method that takes 20 ms to execute and that method is executed 1 million times during your program. That 20 ms doesn't sound like a lot, but you're doing it so many times—your program is spending 20 million ms or about 5.5 hours executing the method. If you were to reduce that 20 ms to 15 ms (4.1 hours) or 10 ms (2.7 hours), you would make a huge difference in runtime.

When a technology is brand new to the team, there is a feeling-out period. What can this technology do? How hard can I push it? What are its limitations? During this period, the team will try to see what cases a technology can be used for and how.

It's during this learning period that I get asked very unbounded questions. Can I use technology X do to thing Y? Sometimes the question is perfectly suited to the technology. Other times, it's really stretching what the technology should be used for and how. I give students a two-question framework to help them figure it out: Is it technically possible? Should you do it?

"Is it technically possible?" means, can the technology actually do what you are

asking? That's usually a quicker pass/fail when you know and understand the technology. "Should you do it?" means, is it wise operationally, technically, or code-wise to use the technology like that? In other words, yes, that idea will work, but it will be an absolute abomination and I'm going to have to take your engineering card away. Or, yes, you could write that code, but your operations person will never speak to you again.

Speaking of abominations, sometimes Big Data is abused with small-data use cases. For these times, I give students another two-question framework: Is it Big Data? Will it become Big Data? Using small data solutions leads to all sorts of scaling headaches. What happens when you use Big Data solutions for small data? Your capital expenditures go up, your personnel costs go up, and your project times go up. Those are just some of the high points. It is such massive overkill if it isn't Big Data. This isn't the time to pad your resume.

Building Data Pipelines

The primary goal of a data engineering team is to build data pipelines. A data pipeline is how your data sources move through your system. It can be real time, batch, or a mix of both. Part of a data pipeline's job is to make the data ready to be processed and then expose that data.

Data pipelines are not made up of a single technology. A Software Engineer who knows a single Big Data technology is not a qualified Data Engineer. You need to use many different technologies to create a solution. This is part of what makes Big Data so complex. A data pipeline needs to bring together 10 to 30 different technologies to do the job, whereas a small data solution will only need 1 to 3.

When we add in NoSQL databases, the situation becomes even more complex. Most organizations think in terms of a single NoSQL database for all of their data. For large organizations with diverse use cases, you'll probably need several different databases that are specific to the use cases you have. Yes, that will increase operational expenses and the amount of data you're storing. However, that will make the use cases perform faster or even enable what you're trying to accomplish.

Knowledge of Use Case

An in-depth knowledge of the use case is one of the key differences between Big Data and small data projects. This is one of the primary ways I've seen data engineering teams fail. They're creating a data pipeline for a use case they don't understand, don't fully understand, or is changing on them.

Starting the creation of the data pipeline before you understand the use case leads you to waste time and money. You'll spend a month writing code that doesn't get used.

What happens when you don't have a use case

A large retail company started its Big Data project without a use case. They already had a data warehouse and a data warehousing team in place. They thought that going to Big Data was just a matter of moving from technology A to technology B, but their move to Big Data would require more than just switching technologies.

They also thought they would find a use case along the way, as they had done with small data. By not knowing the use case, they made all the wrong choices on technologies. By not having clear-cut goals, they accomplished none of their goals. The project was eventually shut down.

With small data, you can use virtually whichever technology you already use. You come in with a technology stack and use it for everything. For Big Data, almost every single engineering decision comes down to the use case. If you don't know your use case, you won't be able to make these decisions. When I teach data engineering teams, I tell them not to even start looking at technologies until they've clearly identified the use case.

Choosing technologies first and then looking at use cases often leads to failure. There is a wide spectrum of possible levels of failure. The best-case failure is that the project takes longer or much longer to complete. The worst-case failure is that the use case isn't possible at all with the technology choices. I've seen all levels in the spectrum, from teams who've lost a million dollars and six months by using a technology incorrectly, to those who lost a month and $150,000 by

not understanding the technology's limitations.

You've probably heard the saying that an ounce of prevention is worth a pound of cure. With Big Data solutions, a pound of prevention is worth a ton of cure. Yes, you will spend far more time understanding the use case, but the payoff is huge.

Another common mistake is to treat the design of a Big Data solution as you would for a relational database. When you design for a relational database, you're thinking about how data (tables) relate to each other and how to normalize that data. When you design for Big Data, you're thinking about use case and how data is accessed, processed, and consumed. You are entering a very different mindset and engineering focus.

Failing to think about use case when designing your data layout also leads to poorly performing data pipelines.

It's worth noting that use cases can grow if a business had to make concessions on its original goals due to technical limitations. For example, the business originally wanted to process 20 years of data and two datasets, but technical limitations only permitted 5 years of data and a single dataset to be processed. Once you have Big Data technologies in place, those limitations are removed. The business will want to go back to their original ask and the use case will grow.

Another way the use case will grow is when other members of the organization hear about the capabilities and want to use them, too. This happens when I'm training a team. They'll get emails from other teams that have heard they're learning a Big Data technology and now want to start using it themselves.

Right Tool for the Job

A major portion of a qualified Data Engineer's value is to choose the right tool for the job. When creating a Big Data solution, they must choose from a bevy of tools and technologies that perform similar tasks while offering different tradeoffs and abilities. For qualified Data Engineers, the choice comes down to knowing the use case.

Another value add of a qualified Data Engineer is to help figure out if a solution requires Big Data or small data technologies. Spend the money and time to hire a qualified Data Engineer and save yourself some heartache. In the best case,

using the wrong tool costs you hours or weeks of time; in the worst case, you waste months before learning the task is impossible to complete with that tool.

I'm trying to keep this book technology free, but I want to give a common example I teach teams. Let's say you need to process some data. In this use case, you could use a Big Data SQL engine or hand-code a program. Which one should you choose? Consider these questions:

- How often will the program run? Is this a one-off call to a program or is it run hourly or daily?
- If you were to hand-code it, what level of performance improvement would you see, if any?
- Is there a time when the SQL would become so complex that you'd be better served hand-coding?

These are the sorts of questions I go through with data engineering teams. Sometimes engineers think they're selling themselves short or not showing their full skills by using a SQL technology. I explain that there's no shame in using SQL as long as it does the job. Use the time it saved you to improve another part of the pipeline.

In this example, depending on complexity of the query/code, it would save or lose a few hours of time. That's obviously not a make-or-break decision for the team. By compounding many decisions like this one, you get a significant time savings. In more extreme examples, using the wrong technology for the job means the team loses six months of work.

Proper training also helps the data engineering team choose the right tool for the job. Big Data technologies are constantly changing and improving. New technologies are being added to address niches in the market.

There's a certain sense of bravado that I run into. I'll teach a team that tells me they're advanced and already know the technology. They don't need any help. There are a few outliers that do understand the technology and don't need any help, but the vast majority doesn't; they need some help and training. There is no shame in getting coaching or training on a brand-new technology. Nor is it an admission that you're not intelligent. A smart data engineering team knows that early is the best time to get help.

I see this all the time at companies who start coding before I come in to train them. The average amount wasted before training is $100,000 to $200,000. I go

through the use case with the team and make sure the technology and designs are correct. I stop teams from putting designs and systems in production where it would have cost $500,000 to $1 million in operational and development costs. The worst I have seen is a team wasting $1 million to $1.5 million and six months of development time.

A big reason for wasted time and money are the subtle nuances to the Big Data technologies. It's hard for a Data Engineer to absorb these nuances just by picking up a book. Sometimes these subtleties are around use cases, inherent in the technology, or implicit assumptions in the technology. Books like this one are general in nature. I can't talk about every team's use case because that isn't practical and I don't want to bore you to death.

Subtle technological nuances may, in light of your use case, make something either difficult or impossible. When I said Data Engineers need to know 10 to 30 different technologies, this is why. The Data Engineer needs to know these subtleties and then figure out what the best tool for the job is. It bears repeating: that tool may or may not be part your company's current stack.

CHAPTER 7

Running Successful Data Engineering Projects

Crawl, Walk, Run

If your data engineering team is moving from small data to Big Data, you'll need to set them up for success. Going from zero to Big Data is a common way that Big Data projects fail; you'll need to get them going in a less jarring way. Attacking a Big Data project with an all-or-nothing mindset leads to an absolute failure. I highly suggest breaking the overall project into more manageable phases. These phases are called *crawl*, *walk*, and *run*.

If you go directly from crawling to running, you'll trip and fall. You'll continually trip and fall and not know why. This is a common issue with new data engineering teams. They don't set up a good foundation before they move on to the advanced phases.

I'm going to talk about what each phase is, offer some suggestions on what it should include, and discuss what your focus in each should be.

Crawling

The crawling phase is doing the absolute minimum to start using Big Data. This might be as a simple as getting your current data into a Big Data technology. For highly political organizations, this is the phase where you try to get groups to start working together. Ideally, you're getting other groups that are opposed to Big Data to buy into the value.

A good stretch goal for this phase is to automate the movement of data to and from your data pipeline. You'll want to continually bring in data.

In this phase, you'll start on your ETL coding and data normalization. This will set you up for success as you start analyzing the data. This phase has minimal

amounts of analysis. Your focus is on creating the system that will make it as simple as possible to analyze the data.

Walking

The walking phase builds on the solid foundation you laid while crawling. Everything is ready for you to start gaining value from your data. If you didn't build a solid foundation, everything will come crashing down.

In this phase, you're starting to really analyze your data. You're using the best tool for the analyses because the cluster is already running the right tools.

You're also creating data products. These aren't just entries in a database table; you are focusing on in-depth analysis that creates business value. At this point, you should be creating direct and quantifiable business value.

Last, you're starting to look at data science and machine learning as ways to improve your analysis or data products.

Running

In the running phase, you're moving into the advanced parts of Big Data architectures. You're gaining the maximum amount of value from your data.

You're also looking at how batch-oriented systems are holding you back, and start looking at real-time systems.

You're focusing on the finer points of your data pipeline. This includes looking at how to optimize your storage and retrieval of data. This might include choosing a better storage format or working with a NoSQL database. You're also using machine learning and data science to their fullest potential.

Iterating

The next question is how to maintain this velocity. Data pipelines aren't really ever done. They're constantly changing as you add new datasets and technologies, and other teams start using the data.

The team needs to iterate and repeat the crawl, walk, run again. This time, the crawl may be not as simple or as time consuming; it could be more around validating that the task is possible. The walk and run phases would be mostly the same.

Technologies

This book is not about specific Big Data technologies. They will come and go. However, the general patterns will stay the same. The subtle nuances will change. The tradeoffs of one technology or another will change.

Technologies should only be chosen after looking at the use case and verifying they will work for it. This is a common issue where companies zero in on a specific technology. They're used to small data solutions where everything can be accomplished with the same stack. That isn't the case with Big Data. A Big Data solution is completely geared around the use case.

The Data Engineers on the team will need to keep up with the rapid change of technology, especially in real-time systems. There are many different technologies to keep up with. Only by specializing in Big Data can the Data Engineers stay current with everything that's happening.

There is a tendency to blame technology for project or data pipeline failures. The technologies work just fine. It's far more difficult to look inwardly at a team and company to understand why they failed. There's usually several pieces and reasons, but technology isn't one of them—people and process are at the top of the list of why teams fail. The team may not have followed the steps I outline in this book and therefore failed.

The focus on specific technologies often comes from the technology vendors themselves. They are pushing their own solution as the solution to everything. In my experience, companies need a wide range of solutions and they'll need them from several different companies.

Why Do Teams Fail?

Part of being successful in a project is to learn from others' failures. I'm writing this book partly because I'm tired of seeing teams repeat the same pattern of failure. Most teams fail for the same or similar reasons, be it one reason or a combination of several.

When you're creating small data solutions, you have a much lower technical bar and level of complexity. If the data engineering team couldn't do something well, it could be covered over by other means. With Big Data, anything you don't do well with small data just gets magnified. For example, if the team can't handle creating or using complex systems, a Big Data solution will just magnify

this deficiency. And as noted, data pipelines are solutions that include 10 to 30 complex technologies.

A very common recipe for failure is when the entire data engineering team is made up of DBAs or Data Warehouse Engineers. The data engineering team needs to be multidisciplinary, so you will need to check for skills gaps.

Related to the skills gaps is when a team doesn't program or have Java experience. Big Data technologies require programming and Java is their predominant language. Not having any programming skills on the team is a big red flag. Not knowing Java (or Python or Scala) is an issue, but not a showstopper. Programmers in one language can learn enough Java to get by. However, you do need to give the team the resources and time to be successful with a brand-new language. You'll also need to verify that any project timelines take into account the team having to learn and program in a different language.

Other teams lack a clear use case. They've been told to go find one or Big Data is seen as a silver bullet that will solve any problem. Without a clear use case, technologies cannot be chosen, solutions written, or value created. These projects are doomed because there is no finish line.

Other teams face an albatross in the form of extensive legacy systems. The new data pipeline is expected to be backward compatible or support all of the previous systems. This stymies execution, leading to failure when projects' plans and managers don't account for the increase in time. Heavy legacy systems integrated with data pipelines often take 50% longer to complete.

Perhaps my biggest pet peeve when working with data engineering teams is when they're set up for failure. There's little to no preparation on the project. The engineering team doesn't have a use case and hasn't been given the resources to figure it out. I hate seeing doomed projects.

Other teams lack qualified Data Engineers or don't have a project veteran. These teams may be initially successful, but over the long term they will fail. They'll get stuck at the crawl phase. They won't have the skills or abilities to progress beyond the basics. A project veteran can help lead the team through the difficult transitions of the walk and run phases.

Some teams lack members who understand the entire system and the importance of schema. These failures don't happen initially; they manifest in the walk and run phases, when other parts of the organization start using the data

pipeline. The team who didn't engineer their schema properly will end up with a maintenance nightmare on their hands. A change to a file will have a ripple effect on the rest of the data pipeline.

Some organizations think Big Data is simple and cheap. These are the organizations that fail to pay Data Engineers sufficiently and don't get qualified ones. The organization fails to give the team the resources they need to be successful and then holds them accountable for failing. They think a Big Data project can be done in the same timeframes as a small data project. They're wrong, and they'll fail over and over until they make changes.

Still other organizations are incredibly ambitious. They'll show me their use case and data pipeline. It will be the holy grail of data pipelines. I'll start to ask them questions about their data engineering team and their experience, and discover the team will be absolute beginners. This mismatch of ambition and actual skill causes failures.

Why do companies fail?

Sometimes it's the entire company that causes the failure. The data team could be productive and provide a data pipeline that doesn't get used. A company needs to a have a data-driven culture and data-centric attitude. Without this attitude, the data engineering team's insights and data won't be transformed into business value. Other times, you need to make changes to the other parts of the organization to fully leverage your data.

Why Do Teams Succeed?

In my experience, a team's success starts and ends with the people on it. If you have the wrong people, the project won't go anywhere. If you have qualified people who work hard, you will have a good chance at success.

Getting training is the biggest key to success I've seen. A team who has been trained on a technology has a high chance of success. They've been able to ask the right questions and compress the time needed to the learn the technology. I compare this to questions I get and see from untrained people. The question itself often shows a general misunderstanding of the technology itself. They

could receive the correct answer, but they won't be able to apply it due to their lack of understanding. Yes, there are people who can learn on their own. They're few and far between and they're definitely outliers.

It's never too late to turn the ship around

I taught at a large media and digital advertising company. As I was preparing to go there, the salesperson told me that the team was six months into the project and their CxO and VPs were closely watching this project. I thought that was very odd. Usually a team will train early on in the project or not at all.

I arrived to start the first day of class. I talked to the class about the project and what was going on. The team was indeed six months into the project and it was woefully behind schedule. The team really had their backs to the wall.

A few things were clear about the team. They were smart, dedicated, and working hard. It was also clear they had fundamental misunderstandings of the technology and how it should be used. I explained to the class that I would be teaching the class differently than I normally do. I would be going deeper into the things they didn't understand.

I also told them to schedule a meeting for the following Monday. I told them the meeting should take four to six hours because they're going to have to assess how much time it will take to rewrite their software once they correctly understand the technology.

Throughout the class I would say, "Take note of that. That's something big you're going to have to change. Make sure to talk about it on Monday."

I kept up with the team after that class. I wanted to know how the project turned around, and it did. The team turned itself around after a month and shipped the product.

Before I arrived, the team had wasted six months and $1 million to $1.5 million. Worse yet, the team's CxO and VPs weren't happy. After the team turned itself around, they became the toast of the company. Many of the members were promoted based on their ability to create successful Big Data projects. They could have saved themselves time and money by getting trained from the outset.

Another characteristic is that the team is set up for success. The organization hasn't just thrown the team into a no-win situation. They've given the team the resources and time to complete the tasks.

A final characteristic is that the organization has rational expectations for team. The team actually has an achievable and known finish line. Now, there will be many different races over a project's life, but each one should have an understood and reasonable finish line. There is nothing more depressing and morale shattering than going into a project you know the team can't finish. A reasonable, achievable goal gives the team confidence and greater velocity in finishing the project or task.

Paying the Piper

Some teams' productivity is hampered by long delays in moving to Big Data. In these situations, the company has been experiencing Big Data problems for some time. Instead of actually solving the problem, they've been patching and propping up small data solutions as long as they can. This leads to a massive technical debt.

I've seen and experienced these delays before. I've worked at companies where they're investing in propping up a small data solution by shoring up their database. They had a DBA dedicated just to looking for queries to optimize and any other scaling issues. That came at the expense of actually putting money toward a real solution to a Big Data problem.

I've taught at other companies doing a similar thing. They're too busy shoring up their small data solution that they can't look far enough into the future to fix it with a Big Data solution.

I tell these companies they're a train speeding toward a reinforced concrete wall. It may be a year, two years, or five years before they hit that wall, but they will. If you can do a back-of-envelope calculation of how fast your data is growing, you can calculate when you'll hit that wall.

Let me give you a common and concrete example. Financial organizations often need to run end-of-day reports. These are time boxed and absolutely have to be done at a certain time or there's a monetary penalty. Let's say you have five hours to run the job and it's taking four hours to complete. The team is putting their finger in the dike by vertically scaling the processing and trying to optimize

the database. Doing your back-of-envelope calculations, you see that you'll hit the wall in four years.

You breathe a sigh of relief because four years is a long time away. Except:

- The organization could grow much faster than you expected
- The organization could purchase another organization and need capacity sooner
- Given the legacy systems and complexity, it could take two to four years to finish a solution

If you have Big Data problems that you're attempting to solve with small data solutions, you'll eventually have to pay the piper. Doing so too late could mean crashing into a reinforced concrete wall of catastrophic failure.

Some Technologies Are Just Dead Ends

Some teams thought they would never have to rewrite their pipeline. They could be early adopters of a technology and chose what they believed were the winners. Some of these Big Data technologies range from graduate school thesis code, to code written in an enterprise that was donated to open source. The varying degrees of completeness and enterprise focus of these technologies contribute to their demise.

This may surprise you, but not all graduate school thesis code is well thought out or production ready. Unless you or your team has the ability or resources to evaluate technology at this level, you're going to have to rewrite.

Sometimes the vendors have the right idea and the wrong implementation. Let's face it, the vendors aren't immune from advancing their own agendas and policies. This is when the worst abominations are born. Other times, the vendors have good ideas but the business just doesn't invest in the technology.

If you imagine a spectrum with conservative bets on one end and aggressive bets on the other, that's how you can think about Big Data technologies. The more aggressive, the higher the odds of wholesale rewrites and technology changes.

Some technologies pride themselves on how much they change. If they didn't get it right two years ago, what makes you think they're going to get it right next time?

To that end, you'll want to take steps to mitigate your risks. One simple step is to use conservative, already proven technologies. Conservative doesn't mean cutting edge; sometimes you'll need cutting-edge technology. Some technologies are written to keep you from writing directly to their APIs. By not writing directly to the technology's API, a team can move code around without having to rewrite it. Apache Beam is one such technology.

What if You Have Gaps and Still Have to Do It?

Sometimes I'll teach at a company where the team is wholly unprepared for the tasks ahead of them. At some companies, the gap is small and just needs to be traversed. At other companies, the gap is a canyon, and crossing it will require a significant amount of help.

The first step is to identify which skills were missing and how important those gaps are to the team. This step requires a great deal of honesty.

Let's go through a few common examples I've seen with data engineering teams.

No Programming Skills

The most common scenario I've seen are data engineering teams with no programming skills. The team is replete with DBAs and SQL skills, but lack the ability to program the complex systems. This team makeup often comes from taking data warehousing teams and calling them data engineering teams.

At a large organization, I'll look around for data-focused programming teams. Failing that, I search for programming teams with multithreading skills. If I still can't find that team, I look for any Java or Python team, with a preference for Java.

Each of these teams will need training; it's just a matter of how much they'll need. There's also gradually rising odds on these teams having an ability gap to contend with. Not all people with programming skills can handle the complexity of Big Data.

If you can't or don't have any programming skills in the company, you'll need to hire one or more qualified Data Engineers as a consultant. You can also hire a company to write the code for you. Here is where I urge caution, as some consulting companies will say they do Big Data solutions but are just as lost as

their clients. Look for consulting companies with a proven track record in Big Data projects.

Actual Programming Skills

I mentored a large financial company who didn't have the programming skills to create the data pipeline. They decided to hire a consulting company before I started helping the team.

During the kickoff meeting, I started asking the rudimentary questions about the technology choices and the use cases because the consulting company had already started the project. They couldn't answer any of my foundational questions. Confused, I asked the team if someone else was tasked with the code and if I was talking to the wrong person—I was talking to the right person.

It turned out that the team hired a consulting company who only did data warehousing. They didn't have the programming skills, or ability, to create the solution. The consultants were hoping I'd train them and show them how to create the solution. I had to tell the company they made a mistake in hiring the consulting company and they needed to fire them.

No Project Veteran

Hiring a project veteran can be tricky. One may not be available, or you may not be able to afford one. You'll have to contend with your less experienced team.

When no project veteran is present, I keep a sharp eye out for abuses of the technology. I also keep an eye out for the team missing the subtle nuances of the use case and technology. These teams tend to create solutions held together with more duct tape and hope than you'd like to see.

Absent a veteran, you can get the team mentoring to help teach them over time. You can get training; in my training classes, we go through at least one use case. You can also get specific consulting. This could range from a second pair of eyes to go through your design, all the way to having a long-term relationship with a consultant.

Lack of Domain Knowledge

Another scenario I've seen is where a data engineering team is brand new and hired from outside of the company. A lack of understanding of the domain can lead to a misunderstanding of the use case. That can spell disaster for the project.

This scenario happens most often when a data engineering team is replacing a legacy system. The team that created, supports, and makes their living off the legacy system feels threatened. I've trained at many companies where this is the case.

I've helped them by training both teams at the same time. I'll have members of the legacy and data engineering teams attend so that they're getting the same knowledge. The Trojan horse is that I'm breaking down the political barriers between the teams. This may surprise you, but sometimes companies don't communicate effectively internally. The training session is a great time to handle this breakdown.

I suggest you handle this lack of skill politically rather than through business or technical means. By breaking down the political walls, you score allies from the legacy team to help you. You'll learn how and why they engineered the solution the way they did. You'll learn some of the pitfalls they hit while creating.

By solving this problem, you'll gain the legacy team's domain knowledge for your team.

CHAPTER 8

Steps to Creating Big Data Solutions

I hope you didn't just skip to this chapter thinking you didn't need to read the rest of the book. You won't understand the whys and hows without having read it. That's a good way to make your Big Data solution fail.

These steps are at the 30,000-foot level. These are not the granular day-to-day steps. Those steps are specific to your development methodology like Waterfall or Agile.

Agile Data Engineering Teams

The most productive data engineering teams are using Agile methodologies. If you are still using Waterfall and are starting with Big Data, you'll need to switch. You need to respond to new issues with more speed than Waterfall allows. Big Data is too complex to plan everything ahead of time.

Pre-project Steps

Before you start on your Big Data project, you'll need to take care of some tasks on creating the team.

Create a Data Engineering Team

If you don't already have a team in place, you will need to create one. Often companies don't understand the need to create an actual data engineering team. I've already outlined why this should be done earlier in the book in Chapter 2, "The Need for Data Engineering," and Chapter 3, "Data Engineering Teams."

Check for Gaps

If you already have a team or are in the process of creating it, you will need to check for gaps.

The first gaps to check for are skills gaps. Use the skills gap analysis mentioned earlier in the book. This requires enormous honesty about the team's skills.

The next gap to check for is an ability gap. You may have people on the team who will never be able to perform. The significant uptick in complexity puts Big Data out of their reach and no amount of time and help is going to change that. It would be unfair to expect people with an ability gap to perform on a team.

You didn't skip one of the most crucial parts?

If you didn't perform the skills gap and ability gap checks in Chapter 3, "Data Engineering Teams," you need to. This exercise isn't optional.

Use Case

A major difference between Big Data and small data projects is how deeply you need to know and understand your use case. Skipping this step leads to failed projects. Every decision after this is viewed through the lens of the use case. Furthermore, there may be many different use cases, and you will need to understand each one.

Questions you should ask about your use case include the following:

- What are you trying to accomplish?
- How fast do you need the results? Do they need to be in real-time, batch, or both?
- What is the business value of this project?
- What difference will it make in the company?
- In what time frame does the project need to be completed?
- How much technical debt do you have?
- How many different use cases are there?
- How much data will we need to store initially? How much will we need to grow by on a daily basis?

- Will we need to store a wide variety of data types?
- How big is our user base? If you don't have users, how many sources of data or devices will you have?
- How computationally complex are the algorithms you're going to run?
- How can you break up your system into crawl, walk, run?
- Is the use case changing often?
- How is data being accessed for the use case?
- How secure and encrypted does the data need to be?
- How will you handle the data management and governance?
- Who will own the data and are they encouraging other teams to use it?

Teaching and Use Cases

When I teach, I have to make an effort to learn the organization's use case. This is because the answer to almost all of their questions is use case dependent. Whenever they ask me a question, the answer isn't a one-size-fits-all response. It comes down to what they're doing and how they're doing it. Only by knowing their use case can I remotely answer their question.

Is It Big Data?

Now that you understand your use case, you'll need to figure out if it requires Big Data technologies. This is where a qualified Data Engineer is crucial. Given your knowledge of the use case, the Data Engineer will help you make that decision.

A common word to look for when compiling the use case information is "can't." When you're talking to the other team members about the use case, they'll often talk about how they're trying to do this now or they've tried it before. For both of these scenarios, they "can't" because of some technical limitation related to using small data technologies for Big Data problems. By changing the technologies to Big Data ones, the answer will change to "can."

Will It Become Big Data?

Sometimes an organization will not have or be experiencing Big Data problems yet, but they're projecting Big Data problems in the near future. These are organizations like the ones facing a train hitting the reinforced concrete wall. Or

they're startups that are expecting to grow exponentially. Either way, they are an organization that doesn't have Big Data now, but will someday.

They're faced with the difficult decision of when to start using Big Data technologies. I've seen what happens in both cases.

For startups, I've seen them use small data technologies and hit it big. Then the startup can't move over to a Big Data technology fast enough and it starts losing customers and traffic. I've also seen startups make the deliberate choice not to implement Big Data technologies due to the complexity and increased project times. Those companies ended up going out of business before they needed Big Data technologies.

For larger organizations, I've seen them delay moving to Big Data technologies due to their legacy systems. They create so much technical debt for themselves that they spend years moving to a new system. They just have to hope that they don't hit a reinforced concrete wall during that time.

Evaluate the Team

Now that you have decided that your use case must be solved with one or more Big Data technologies, which are correct and why? This portion requires an honest and realistic look at th team's makeup and abilities. To determine if your team can execute a Big Data project, ask yourself:

- Does my team have a background in distributed systems?
- Does my team have a background in multithreading?
- What programming language(s) does my team use?
- What small data technologies is my team already familiar with?
- Some teams have an unrealistic view of their abilities. Does your team suffer from this?
- Do you and the team realize the increase in complexity associated with Big Data?

Training, Mentoring, Consulting

After a very honest look at the team, you will need to see how you can set the team up for success. Failing to set the team up for success is a common way to make projects fail.

In the "How I Evaluate Teams" section of Chapter 5, I talk about how to figure out what level of help a team needs. An average data engineering team will need training and mentoring. An advanced team may just need training on new technologies.

A team with a very low probability of success will need in-depth training and consulting. After honestly assessing the abilities of the team, you may need to hire a consulting company. This company either could write the portions of the data pipeline that are too difficult for the team or write it entirely.

Choose the Technologies

After your honest and realistic look at the team, you need to decide on technologies. Make this a starting point for discussion with the team. This is another step where it's incredibly important to have qualified Data Engineers. Think about these questions:

- What are the technologies that will make the project successful?
- What are the technologies that my team can be successful with?
- Does your team or company suffer from NIH (not invented here)? The team may want to roll their own or create their own distributed system. (Usually not a good idea.)
- Does the team have the training necessary to be successful?

Write the Code

This is the fun step for data engineering teams—it's where you code out the use case. You start to wire the various pieces and technologies together. It's where you get into the specifics of the Big Data technologies and implement things.

Writing the code is another common place where teams fail. Instead of doing the preceding steps, they go and directly start writing the code. They've already chosen the technology and haven't looked over their team or use case. They're six months into the project and realize they've made a mistake.

Yes, this is the make-or-break step. You find out the hard way if you followed everything I've outlined.

Evaluate

This step is where you look at the data pipeline you've written. Then you compare it against the goals for that phase of the project. You don't just evaluate the project at a technical level. You need to evaluate it at the use case and business levels.

It's incredibly important to evaluate the project, yet it's often not done. In Agile methodology, this would be similar to a retrospective, except that it would be for that part of the project or entire POC. This is an important opportunity for the team to learn from previous issues.

Ask yourself the following questions:

- Did the iteration solve the business need?
- Did the iteration take the amount of time you thought it would?
- How high quality is the code that makes up the iteration?
- Would you actually put this level of code quality in production?
- Was the iteration so simple that it gave the CxOs or upper management a false sense of timelines and complexity?

Repeat

In this step, you figure out how to break the project down into small pieces. This way you're in an iterative approach to creating a data pipeline instead of an all-or-nothing one.

Projects that have an all-or-nothing approach often fail due to timelines and complexity. It is important to break up the overall design or system into phases—crawl, walk, run. I encourage you to segment your development phases to create the system gradually rather than all at once. In approaching this, ask these questions:

- How can you break up your system into crawl, walk, run?
- Has the rest of the management been informed which parts or features of the system will be in which phase?
- Is the next phase in the project vastly more complex or time consuming than the last phase?
- Do the phases have a logical progression?
- Is one feature dependent on a feature down the line?

Probability of Success

When I work with teams, I create a probability of success. This probability takes into account:

- The team's abilities
- The gaps in the team's skills
- The complexity of the use case
- The complexity of the technologies you need to use
- The complexity of the resulting code
- The company's track record on doing complex software
- How well the team is set up for success
- How much external help the team is going to receive (training, mentoring, consulting)

In my experience, these have ranged from a 1% chance of success to a 99% chance.

Next, calculate your probability of success. This requires an incredibly honest look at things. As you looked over the questions and topics in this book:

- Did you have deep reservations about the team succeeding?
- Did they remind you of your team and their skills as they currently stand?

Now, I want you to think of ways to increase that probability:

- Would more qualified Data Engineers increase it?
- Would getting more or better external help increase it?
- Do you need to vastly decrease the scope or complexity of the use case?
- Is there a gap, in skill or ability, that's making success improbable?

If you have a high probability, congratulations. You'll likely be successful. However, many teams fool themselves with a dishonest assessment of their probability of success. They significantly overestimate their abilities.

If you have a low probability, you should really think hard about undertaking the project. This is a time to get extensive external help. At a minimum, your team will need training and mentorship. If it's very low, you should honestly consider having a competent consulting company handle the entire project. Without doing that, you'll be setting the team up for failure, which isn't fair to you or the team.

CHAPTER 9

Conclusion

Best Efforts

Big Data systems are incredibly complex. Part of your job will be to educate your coworkers of this fact. Failing to do so will make everyone compare your progress to easier projects, like a mobile or web project.

Despite your best efforts and planning, Big Data solutions are still hard. When you start falling behind in the project or hitting a roadblock, I highly suggest getting help quickly. I've helped companies who had been stuck for several months. Had they reached out for help sooner, they could have saved months of time and money. This isn't admitting failure or an issue with the team; Big Data is really that hard.

Best of luck on your Big Data journey.

About the Author

Jesse Anderson is a Data Engineer, Creative Engineer and Managing Director of Big Data Institute.

He trains and mentors companies ranging from startups to Fortune 100 companies on Big Data. This includes training on current edge technologies and Big Data management techniques. He's mentored hundreds of companies on their Big Data journeys. He has taught thousands of students the skills to become Data Engineers.

He is widely regarded as an expert in the field and his novel teaching practices. Jesse is published on O'Reilly and Pragmatic Programmers. He has been covered in prestigious publications such as The Wall Street Journal, CNN, BBC, NPR, Engadget, and Wired.

Data Engineering Teams

Creating Successful Big Data Teams and Products

Get expert guidance on how to create, staff, and run your data engineering team. While many sources say that you need a Big Data strategy, few of them talk about how to run a successful Big Data project or create a team that is capable of getting a product into production.

This book takes years of mentoring, teaching, and consulting with teams all over the world and distills it down to an approachable book. It is a practical and field tested guide to making your team successful with Big Data projects.

Here is what you'll learn in *Data Engineering Teams*

- Why do some teams succeed and why do some fail?
- Why is it absolutely crucial that data engineering teams understand and flesh out the use cases?
- What are the exact steps to follow when creating and executing a Big Data project?
- Why is Big Data so much more complicated than small data?
- What is your team's probability of success when creating your Big Data project?
- Why do some teams appear stuck and unable to make progress?
- What is a qualified Data Engineer and why is it crucial to have one on your team?
- When should a data engineering team start looking at and choosing the technology stack?
- What is a data engineering team and which skills does a data engineering team need?

Jesse Anderson is a Data Engineer, Creative Engineer and Managing Director of Big Data Institute.

He works with companies ranging from startups to Fortune 100 companies on Big Data. This includes training on cutting edge technologies like Apache Kafka, Apache Hadoop and Apache Spark. He has taught over 30,000 people the skills to become data engineers.

He is widely regarded as an expert in the field and for his novel teaching practices. Jesse is published on O'Reilly and Pragmatic Programmers. He has been covered in prestigious publications such as The Wall Street Journal, CNN, BBC, NPR, Engadget, and Wired.

ISBN 978-1-7326496-0-6
90000
9 781732 649606

Big Data Institute
www.bigdatainstitute.io