The Transformative Power of Unified Enterprise Data

Machine learning makes unprecedented data unification possible. What could that mean for you?

Find out at www.tamr.com
This Preview Edition of *Getting Data Operations Right*, Chapters 1–3, is a work in progress. The final book is currently scheduled for release in April 2018 and will be available at [oreilly.com](http://oreilly.com) and other retailers once it is published.

*Michael Stonebraker, Nik Bates-Haus, Liam Cleary, Larry Simmons, and Andy Palmer*
# Table of Contents

1. **Introduction** ................................................................. 1  
   - DevOps and DataOps ................................................... 2  
   - The Catalyst for DataOps: “Data Debt” ......................... 2  
   - Paying Down the Data Debt ...................................... 3  
   - From Data Debt to Data Asset ..................................... 4  
   - DataOps to Drive Repeatability and Value .................... 4  
   - Organizing by Logical Entity ..................................... 5

2. **Moving Towards Scalable Data Unification** ..................... 7  
   - A Brief History of Data Unification Systems ............... 7  
   - Unifying Data .......................................................... 8  
   - Rules for scalable data unification ............................ 11

3. **DataOps as a Discipline** ............................................... 13  
   - Why DataOps? .......................................................... 13  
   - Agile Engineering for Data and Software .................... 14  
   - The Agile Manifesto .................................................. 15  
   - Agile Practices ....................................................... 17  
   - Agile Operations for Data and Software ..................... 18  
   - DataOps Challenges .................................................. 23  
   - The Agile Data Organization ...................................... 26
Over the past three decades, as an enterprise CIO and a provider of third-party enterprise software, I’ve witnessed first-hand a long series of large-scale information technology transformations, including Client/Server, Web 1.0, Web 2.0, Cloud and Big Data. One of the most important but underappreciated of these transformations is the astonishing emergence of DevOps.

DevOps—the ultimate pragmatic evolution of agile methods—has enabled digital-native companies (Amazon, Google, etc.) to devour entire industries through rapid feature velocity and rapid pace of change, and is one of the key tools being used to realize Marc Andreessen’s portent that “Software is Eating the World.” Traditional enterprises, intent on competing with digital-native internet companies, have already begun to adopt DevOps at scale. While running software and data engineering at the Novartis Institute of Biomedical Research, I introduced DevOps into the organization, and the impact was dramatic.

Fundamental changes, such as the adoption of DevOps, tend to be embraced by large enterprises once new technologies have matured to a point when the benefits are broadly understood, the cost and lock-in of legacy/incumbent enterprise vendors becomes insufferable and core standards emerge through a critical mass of adoption. We are witnessing the beginning of another fundamental change in enterprise tech called “DataOps”—which will allow enterprises to rapidly and repeatedly engineer mission-ready data from all of the data sources across an enterprise.
**DevOps and DataOps**

Much like DevOps in the enterprise, the emergence of enterprise DataOps mimics the practices of modern data management at large internet companies over the past 10 years. Employees of large internet companies leverage their company’s data as company asset, and leaders in traditional companies have recently developed this same appetite to leverage data to compete. But most large enterprises are unprepared, often because of behavioral norms (like territorial data hoarding), and because they lag in their technical capabilities (often stuck with cumbersome ETL and MDM systems). The necessity of DataOps has emerged as individuals in large traditional enterprises realize that they should be using all the data generated in their company as a strategic asset to make better decisions every day. Ultimately, DataOps is as much about changing people’s relationship to data as it is about technology infrastructure and process.

The engineering framework that DevOps created is a great preparation for DataOps. For most enterprises, many of whom have adopted some form of DevOps for their IT teams, the delivery of high-quality, comprehensive and trusted analytics using data across many data silos will allow them to move quickly to compete over the next 20 years or more. Just like the internet companies needed DevOps to provide a high-quality, consistent framework for feature development, enterprises need a high-quality, consistent framework for rapid data engineering and analytic development.

**The Catalyst for DataOps: “Data Debt”**

DataOps is the logical consequence of three key trends in the enterprise:

1. Multi-billion dollar business process automation initiatives over the past 30+ years that started with back office system automation (accounting, finance, manufacturing, etc.) and swept through the front office (sales, marketing, etc.) in the 1990’s and 2000’s—creating hundreds/thousands of data silos inside of large enterprises.

2. The competitive pressure of digital native companies in traditional industries.
3. The opportunity presented by the “democratization of analytics” driven by new products and companies that enabled broad use of analytic/visualization tools such as Spotfire, Tableau and Business Objects.

For traditional Global 2000 enterprises intent on competing with digital natives, these trends have combined to create a major gap between the intensifying demand for analytics among empowered front-line people and the organization’s ability to manage the “data exhaust” from all the silos created by business process automation.

Bridging this gap has been promised before, starting with data warehousing in the 1990’s, data lakes in the 2000’s and decades of other data integration promises from the large enterprise tech vendors. Despite the promises of single vendor data hegemony by the likes of SAP, Oracle, Teradata and IBM, most large enterprises still face the grim reality of intensely fractured data environments. The cost of the resulting data heterogeneity is what we call “data debt.”

Data debt stems naturally from the way that companies do business. Lines of businesses want control and rapid access to their mission-critical data, so they procure their own applications, creating data silos. Managers move talented personnel from project to project, so the data systems owners turn over often. The high historical rate of failure for business intelligence and analytics projects makes companies rightfully wary of game-changing and “boil the ocean” projects that were epitomized by Master Data Management in the 1990’s.

**Paying Down the Data Debt**

Data debt is often acquired by companies when they are running their business as a loosely connected portfolio, with the lines of business making “free rider” decisions about data management. When companies try to create leverage and synergy across their businesses, they recognize their data debt problem and work over-time to fix it. We’ve passed a tipping point where large companies can no longer treat the management of their data as optional based on the whims of line of business managers and their willingness to fund central data initiatives. Instead, it’s finally time for enterprises to tackle their data debt as a strategic competitive imperative. As my friend Tom Davenport describes in his book “Competing on Analytics,” those organizations that are able to make better decisions faster
are going to survive and thrive. Great decision-making and analytics requires great unified data—the central solution to the classic garbage in/garbage out problem.

For organizations that recognize the severity of their data debt problem and determine to tackle it as a strategic imperative, Data Ops enables them to pay down their data debt by rapidly and continuously delivering high-quality, unified data at scale from a wide variety of all enterprise data sources.

**From Data Debt to Data Asset**

By building their data infrastructure from scratch with legions of talented engineers, digital native, data-driven companies like Facebook, Amazon, Netflix and Google have avoided data debt by managing their data as an asset from day one. Their examples of treating data as a competitive asset have provided a model for savvy leaders at traditional companies who are taking on digital transformation while dealing with massive legacy data debt. These leaders now understand that managing their data proactively as an asset is the first, foundational step for their digital transformation—it cannot be a “nice to have” driven by corporate IT. Even for managers who aren’t excited by the possibility of competing with data, the threat of a traditional competitor using their data more effectively, or disruption from data-driven, digital native upstart require that they take proactive steps and begin managing their data seriously.

**DataOps to Drive Repeatability and Value**

Most enterprises have the capability to find, shape and deploy data for any given idiosyncratic use case, and there is an abundance of analyst oriented tools for “wrangling” data from great companies such as Trifacta and Alteryx. Many of the industry-leading executives I work with have commissioned and benefitted from one-and-done analytics or data integration projects. These idiosyncratic approaches to managing data are necessary but not sufficient to solve their broader data debt problem and to enable these companies to compete on analytics.

Next-level leaders who recognize the threat of digital natives are looking to use data aggressively and iteratively to create new value every day as new data becomes available. The biggest challenge
faced in enterprise data is repeatability and scale—being able to find, shape and deploy data reliably with confidence. Also—much like unstructured content on the web—structured data changes over time. The right implementation of DataOps enables your analytics to adapt and change as more data becomes available and existing data is enhanced.

Organizing by Logical Entity

DataOps is the framework that will allow these enterprises to begin their journey towards treating their data as an asset and pay down their data debt. The human behavioral changes and process changes that are required are as important, if not more important, than any bright, shiny new technology. In the best projects I’ve been involved with, the participants realize that their first goal is to organize their data along their key, logical business entities, examples of which include:

- Customers
- Suppliers
- Products
- Research
- Facilities
- Employees
- Parts

Of course, every enterprise and industry has its own collection of key entities. Banks might be interested in entities that allow fraud detection; agricultural firms might care more about climate and crop data. But for every enterprise, understanding these logical entities across many sources of data is key to ensuring reliable analytics. Many DataOps projects start with a single entity for a single use case and then expand; this approach connects the data engineering activities to ROI from either selling more products or saving money through using unified, clean data for a given entity for analytics and decision making. For each of these key entities any Chief Data Officer should be able to answer the fundamental questions:

- What data do we have?
- Where does our data come from?
• Where is our data consumed?

To ensure clean, unified data of these core entities, a key component of DataOps infrastructure is to create a system of reference that maps a company’s data to core logical entities. This unified system of reference should consist of unified attributes constructed from the raw physical attributes across source systems. Managing the pathways between raw, physical attributes, changes to the underlying data, and common operations on that data to shape it into production-readiness for the authoritative system of reference are the core capabilities of DataOps technologies and processes.

This book will get into much more detail on DataOps and the practical steps enterprises have and should take to pay down their own data debt—including behavioral, process as well as technology changes. It will trace the development of DataOps and its roots in DevOps; best practices in building a DataOps ecosystems, and real world examples. I’m excited to a part of this generational change—one which I truly believe will be a key to success for enterprises over the next decade as they strive to compete with their new digital-native competitors.

The challenge for large enterprise with DataOps is that if it doesn’t adopt this new capability quickly, it runs the risk of being left in the proverbial competitive dust.
The early users of data management systems performed business data processing, mostly transactions (updates) and queries on the underlying data sets. These early applications enabled analytics on the current state of the enterprise. About two decades ago enterprises began keeping historical transactional data in what came to be called data warehouses. Such systems enabled performing analytics to find trends over time, e.g. pet rocks are out and barbie dolls are in. Every large enterprise now has a data warehouse, on which business analysts run queries to find useful information.

The concept has been so successful, that enterprises typically now have several-to-many analytical data stores. To perform cross selling, obtaining a single view of a customer or finding the best pricing from many Supplier data stores, it is necessary to perform data unification across a collection of independently constructed data stores.

This chapter discusses the history of data unification and current issues.

A Brief History of Data Unification Systems

The early systems used to integrate data stores were called Extract, Transform and Load (ETL) products. Given the required amount of effort by a skilled programmer, ETL systems typically unified only a handful of data stores, fewer than two dozen in most cases. The bot-
Tlèneck in these systems was the human time required to transform the data into a common format for the destination repository, writing “merge rules” to combine the data sources, and additional rules to decide on the true value for each attribute in each entity. While fine for small operations, like understanding sales and production data at a handful of retail stores or factories, ETL systems failed to scale to large numbers of data stores and/or large numbers of records per store.

The next generation of ETL tools offered increased functionality, such as data cleaning capabilities and adaptors for particular data sources. Like the first generation, these ETL tools were designed for use by computer programmers, who had specialized knowledge. Hence, they did not solve the fundamental scalability bottleneck, the time of a skilled software professional. These ETL tools form the bulk of the unification market today; however, most large enterprises still struggle to curate data from more than a couple dozen sources for any given data unification project. The present state of affairs is an increasing number of data sources which enterprises wish to unify, and a collection of traditional ETL tools that do not scale. The rest of this white paper discusses scalability issues in more detail.

### Unifying Data

The benefits to unifying data sources are obvious. If a category manager at Airbus wants to get the best terms for a part that their line of business (LOB) is buying, that manager will typically only have access to purchasing data from his own LOB. The ability to see what other LOBs are paying for a given part can help that category manager optimize his spend. Added up across all of the parts and suppliers across all Airbus LOBs, these insights represent significant savings. However, that requires integrating the LOB Supplier databases for each LOB. For example, GE has 75 of them, and many large enterprises have several-to-many because every acquisition comes with its own legacy purchasing system. Hence, data unification must be performed at scale, and ETL systems are not up to the challenge.

The best approach to integrating two data sources of twenty records each is probably a whiteboard or paper and pencil. The best approach for integrating twenty data sources of 20,000 records each
might very well be an ETL system and rules based integration approach. However, if GE wishes to unify 75 data sources with 10M total records, neither approach is likely to be successful. A more scalable strategy is required.

Unfortunately, enterprises are typically operating at a large scale, with orders of magnitude more data than ETL tools can manage. Everything from accounting software to factory applications are producing data which yields valuable operational insight to analysts working to improve enterprise efficiency. The easy availability and value of data sources on the Web compounds the scalability challenge.

Moreover, enterprises are not static. For example, even if Airbus had unified all of its purchasing data, the recent acquisition of Bombardier adds another enterprise’s worth of data to the unification problem. Scalable data unification systems must accommodate the reality of shifting data environments.

Let’s go over the core requirements for unifying data sources. There are seven required processes:

1. Extracting data from a data source into a central processing location.
2. Transforming data elements (WA to Washington, for example).
3. Cleaning data, like -99 actually means a null value.
4. Mapping schema to align attributes across source data sets (e.g. your “surname” is my “Last_Name”).
5. Consolidating entities, or clustering all records thought to represent the same entity. For example, are Ronald McDonald and R. MacDonald the same clown?
6. Selecting the “golden value” for each attribute for each clustered entity.
7. Exporting unified data to a destination repository.

Plainly, requirements 2 – 5 are all complicated by scale issues. As the number and variety of data sources grows, the number and variety of required transforms and cleaning routines will increase commensurately, as will the number of attributes and records that need to be processed. Consider, for example, names for a given attribute, phone number:
To consolidate the two CRM sources into the DataLake Schema, you will need to write one mapping: Phone_Number equals Telephone. To standardize the format of the number, you will need to transform two different formats to a third standard one.

Now let's do this for six data sources:

<table>
<thead>
<tr>
<th>Source</th>
<th>Attribute Name</th>
<th>Record Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRM-1</td>
<td>Tel.</td>
<td>(xxx) xxx-xxxx</td>
</tr>
<tr>
<td>CRM-2</td>
<td>Phone_Number</td>
<td>xxxxxxxxxx</td>
</tr>
<tr>
<td>DataLake</td>
<td>Phone_Number</td>
<td>xxx-xxx-xxxx</td>
</tr>
</tbody>
</table>

We now have five different names for the same attributes; and one of these attributes (“Cell”), may require some expertise to correctly map it to “Phone_Number.” We also have four formats for phone numbers, requiring four different transformations into the data lake one. In this simple example, we’ve gone from three rules to unify three data sources to eight when doubling the amount of attributes. Hence, the complexity of the problem is increasing much faster than the number of data sources. Rules are problematic at scale because:

- They are difficult to construct.
- After a few hundred, they surpass the ability of a human to understand them.
- At scale, they outstrip the ability of humans to verify them.

The first and second generations of ETL systems relied on rules. Creating and maintaining rules, in additional to the verification of the results of those rules, constitutes the bulk of the human time required for rules-based ETL approaches. This is an example of why traditional ETL solutions do not scale. Any scalable data unification must obey the tenets discussed in the next section.
Rules for scalable data unification

A scalable approach therefore, must perform the vast majority of its operations automatically (tenet 1). Suppose it would take Airbus 10 years of labor to integrate all of their purchasing systems using a traditional, rules based approach. If one could achieve 95% automation, it would reduce the time-scale of the problem to six months. Automation, in this case, would use statistics and machine learning to make automatic decisions wherever possible, and only involve a human when automatic decisions are not possible. In effect, one must reverse the traditional ETL architecture, whereby a human controls the processing, into one where a computer runs the process using human help when necessary.

For many organizations, the large number of data sources translates into a substantial number of attributes; thousands of data sources can mean tens or hundreds of thousands of attributes. We know from experience that defining a global schema upfront, while tempting, inevitably fails, because these schemas are invalid as soon as requirements change or new data sources are added. Scalable data unification systems should be discovered from the source attributes themselves, rather than defined first. Therefore, scalable data unification must be schema-last (tenet 2).

As mentioned above, ETL systems require computer programmers to do the majority of the work. Business experts are sometimes involved in specifying requirements, but the people who build and maintain the data architecture are also responsible for interpreting the data they are working with. This requires, for example, a data architect to know if “Merck KGaA” is the same customer as “Merck and Co”? Obviously, this requires a business expert. As a result, scalable data unification systems must be collaborative and use domain experts to resolve ambiguity, thereby assisting the computer professionals who run the unification pipeline. (tenet 3).

Taken together, these three tenets lead us to a fourth one, which is rules-based systems will not scale, given the limitations outlined earlier. Only machine learning can scale to the problem sizes found in large enterprises (tenet 4).

However, machine learning-based solutions do have some operational complexities to consider. While a human can look at a set of records and instantly decide they correspond to a single entity. Data
unification systems must do so automatically. Conventional wisdom is to cluster records into a multi-dimensional space formed by the records’ attributes, with a heuristically specified distance function. Records that are close together in this space are probably the same entity. This runs into the classic N**2 clustering problem; and the computational resource required to do operations with complexity N**2 where N is the number of records is often too great. Scalable unification systems must scale out to multiple cores and processors (tenet 5) and must have a parallel algorithm with lower complexity than N**2 (tenet 6).

Given the realities of the enterprise data ecosystem, scalable unification systems need to accommodate data sources that change regularly. While running the entire workflow on all of the data to incorporate changes to a data source can satisfy some business use cases, applications with tighter latency requirements will require a scalable unification system to examine the changed records themselves and perform incremental unification (tenet 7).

Scalable data unification has to be the goal of any enterprise, and that will not be accomplished using traditional ETL systems. It is obviously the foundational task for enterprises looking to gain “business intelligence gold” from across the enormous troughs of enterprise data.
CHAPTER 3

DataOps as a Discipline

Nik Bates-Haus

Why DataOps?

“Why” is best illustrated by example. A bank’s investment research group uses a real estate dataset that combines internal and external data to follow trends in different real estate markets. The analysts use visualization tools such as Tableau, and analysis tools such as R, to identify investment opportunities. When a market suddenly heats up, as western North Dakota did during the oil boom in the twenty teens, the research group wants access to a trusted dataset as quickly as possible, to avoid missing investment opportunities. Because the data is provided by a data engineering team that already understands the domain, the data structure, the data quality requirements, etc., initial data for the new region can be incorporated in a matter of hours or days. Because the data is delivered by an agile DataOps team, the updated data is immediately available in an acceptance testing environment with no IT intervention.

As the analysts start to use the new data, they will inevitably discover issues. Because of the close integration with DataOps and agile process, these issues can be addressed quickly, while the analysts are already assembling their initial results. Once the issues are resolved and the analytics are finalized, the entire new analytics pipeline can be pushed to production, again with no IT intervention. Having the new dashboards available in production with minimal delay ensures that the bank does not miss investment opportunities in this emerging market.
This example highlights the major benefit of DataOps: data-driven aspects of the business can respond rapidly to changing business needs. DataOps, like DevOps, emerges from the recognition that separating the product—production-ready data—from the process that delivers it—operations—impedes quality, timeliness, transparency and agility. The need for DataOps comes about because data consumption has changed dramatically over the past decade. Just as internet applications raised user expectations for usability, availability, and responsiveness of applications, things like Google Knowledge Panel and Wikipedia have dramatically raised user expectations for usability, availability and freshness of data.

What's more, with increased access to very usable self-service data preparation and visualization tools, there are also now many users within the enterprise who are ready and able to prepare data for their own use if official channels are unable to meet their expectations. In combination, these changes have created an environment where continuing with the cost-laden, delay-plagued, opaque operations used to deliver data in the past are no longer acceptable. Taking a cue from DevOps, DataOps looks to combine the production and delivery of data into a single, agile practice that directly supports specific business functions. The ultimate goal is to cost-effectively deliver timely, high-quality data that meets the ever-changing needs of the organization.

In this chapter, we will review the history of DataOps, the problems it is designed to address, the tools and processes it uses, and how organizations can effectively make the transition to and gain the benefits of DataOps.

**Agile Engineering for Data and Software**

Data Operations (DataOps) is a methodology that spans people, processes, tools, and services to enable enterprises to rapidly, repeatedly, and reliably deliver production data from a vast array of enterprise data sources to a vast array of enterprise data consumers.

DataOps builds on many decades of accumulated wisdom in agile process. It is worth taking a moment to highlight some key goals and tenets of agile, how they have been applied to software, and how they can be applied to data. Agile software development arose from the observation that software projects that were run using traditional processes were plagued by:
• High cost of delivery, long time to delivery, and missed deadlines;
• Poor quality, low user satisfaction, and failure to keep pace with ever-changing requirements;
• Lack of transparency into progress towards goals, and schedule unpredictability;
• Anti-scaling in project size, where the cost per feature of large projects is higher than the cost per feature of small projects.
• Anti-scaling in project duration, where the cost of maintenance grows to overwhelm available resources.

In short, the same frustrations that plague so many data delivery projects today.

The Agile Manifesto

In establishing an approach that seeks to address each of these issues, the Agile community introduced several core tenets in an Agile Manifesto:

Manifesto for Agile Software Development

We value:

1. **Individuals and Interactions** over processes and tools
2. **Working Software** over comprehensive documentation
3. **Customer Collaboration** over contract negotiation
4. **Responding to Change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Let's review these briefly, their impact on software development, and the expected impact on data delivery.

---

1 [https://assets.uits.iu.edu/pdf/Agile-Manifesto.pdf](https://assets.uits.iu.edu/pdf/Agile-Manifesto.pdf)
Tenet #2: Working Software

I’ll start with tenet #2, because it really should be tenet #1: the goal of software engineering is to deliver working software. Everything else is secondary. With working software, users can accomplish their goals significantly more readily than they could without the software. This means that the software meets the users’ functional needs, quality needs, availability needs, serviceability needs, etc. Documentation alone doesn't enable users to accomplish their goals. In fact, since this manifesto was written, many software engineering teams seek to adhere to principles of usability and interface design that make documentation unnecessary for most situations.

Similarly, the goal of data engineering is to produce working data; everything else is secondary. With working data, users can accomplish their goals significantly more readily than they could without the data. This means that the data meets the users’ functional needs, quality needs, availability needs, serviceability needs, etc. The corollary about documentation also applies: ideally, data engineering teams will be able to adhere to principles of usability and data design that make documentation unnecessary for most situations.

The other three tenets are in support of that main tenet, that the goal of a software engineering team is to produce working software. They all apply equally well to a data engineering team, whose goal is to produce working data.

Tenet #1: Individuals and Interactions

Software is written by people, not processes or tools. Good processes and tools can support people and help them be more effective, but neither processes nor tools can make mediocre engineers into great engineers. Conversely, poor processes or tools can reduce even the best engineers to mediocrity. The best way to get the most from your team is to support them as people, first, and to bring in tools and process only as necessary to help them be more effective.

Tenet #3: Customer Collaboration

When it comes to requirements, customers are much more likely to “know it when they see it,” than to be able to write it down. When you try to capture these needs up front in a requirements “contract”, customers will push for a very conservative contract to minimize their risk. Building to this contract will be very expensive, and still
unlikely to meet customers’ real needs. The best way to determine whether a product meets your customer’s needs and expectations is to have the customer use the product and give feedback. Even when a product is very incomplete, or even just a mock-up, customers can give invaluable feedback to guide development to meet their needs better. Getting input as early and as often as possible ensures course corrections are as small as possible.

**Tenet #4: Responding to Change**

Change is constant—in requirements, in process, in availability of resources, etc.—and teams that fail to adapt to these changes will not deliver software that works, either not as well as intended, or perhaps not at all. No matter how good a plan is, it cannot anticipate the changes that will happen during execution. Rather than invest heavily in up front planning, it is much better to plan only as much as necessary to ensure that the team is aligned and the goals are reasonable, then measure often to determine whether course correction is necessary. Only by adapting swiftly to change can the cost of adaptation be kept small.

**Agile Practices**

The preceding has described the goal and tenets of Agile, but not what to actually do. There are many variations of Agile process, but they share several core recommendations:

1. **Deliver working software frequently**—in days or weeks, not months or years—adding functionality incrementally until a release is completed;
2. **Get daily feedback from customers**—or customer representatives—on what has been done so far;
3. **Accept changing requirements**, even late in development;
4. **Work in small teams** (3–7 people) of motivated, trusted and empowered individuals, with all the skills required for delivery present on each team;
5. **Keep teams independent**; this means each team’s responsibilities span all domains, including planning, analysis, design, coding, unit testing, acceptance testing, releasing, and building and maintaining tools and infrastructure;
6. **Continually invest in automation** of everything;

7. **Continually invest in improvement** of everything, including process, design, and tools.

These practices have enabled countless engineering teams to deliver timely, high-quality products, many of which we use every day. These same practices are now enabling data engineering teams to deliver the timely, high-quality data that powers applications and analytics. But there is another transition made in the software world that needs to be picked up in the data world. When delivering hosted applications and services, agile software development is not enough. It does little good to rapidly develop a feature, if it then takes weeks or months to deploy it, or if the application is unable to meet availability or other requirements due to inadequacy of the hosting platform. These are operations, and they require a skill set quite distinct from that of software development. The application of agile to operations created DevOps, which exists to ensure that hosted applications and services can not only be developed but also delivered in an agile manner.

**Agile Operations for Data and Software**

Agile removed many barriers internal to the software development process, and enabled teams to deliver production features in days, instead of years. For hosted applications in particular, the follow-on process of getting a feature deployed retained many of the same problems that Agile intended to address. Bringing development and operations into the same process, and often the same team, can reduce time-to-delivery down to hours or minutes. The principle has been extended to operations for non-hosted applications as well, with similar effect. This is the core of DevOps.

The problems that DevOps intends to address look very similar to those targeted by Agile Software Development:

- Improved deployment frequency;
- Faster time to market;
- Lower failure rate of new releases;
- Shortened lead time between fixes;
• Faster mean time to recovery (in the event of a new release crashing or otherwise disabling the current system).

Most of these can be summarized as availability—making sure that the latest working software is consistently available for use. In order to determine whether a process or organization is improving availability, you need something more transparent than percent uptime, that can be measured continuously and tells you when you’re close, and when you’re deviating. Google’s Site Reliability Engineering team did some of the pioneering work looking at how to measure availability in this way, and distilled it into the measure of the fraction of requests that are successful. DevOps, then, has the goal of maximizing the fraction of requests that are successful, at minimum cost.

For an application or service, a request can be logging in, opening a page, performing a search, etc. For data, a request can be a query, an update, a schema change, etc. These requests might come directly from users, e.g. on an analysis team, or could be made by applications or automated scripts. Data development produces high-quality data, while DataOps ensures that the data is consistently available, maximizing the fraction of requests that are successful.

**DataOps Tenets**

DataOps is an emerging field, whereas DevOps has been put into practice for many years now. We can use our depth of experience with DevOps to provide a guide for the developing practice of DataOps. There are many variations in DevOps, but they share a collection of core tenets:

1. Think services, not servers
2. Infrastructure as code
3. Automate everything

Let’s review these briefly, how they impact service availability, and expected impact on data availability.

---

Tenet #1: Think Services, not Servers

When it comes to availability, there are many more options for making a service available than there are for making a server available. By abstracting services from servers, we open up possibilities such as replication, elasticity, failover, etc., each of which can enable a service to successfully handle requests under conditions where an individual server would not be—for example, under a sudden surge in load, or requests that come from broad geographic distribution.

This should make it clear why it is so important to think of data availability not as database server availability, but as the availability of Data as a Service (DaaS). The goal of the data organization is not to deliver a database, or a data-powered application, but the data itself, in a usable form. In this model, data is typically not delivered in a single form factor, but simultaneously in multiple form factors to meet the needs of different clients: RESTful web services to meet the needs of service-oriented applications, streams to meet the need of real-time dashboards and operations, and bulk data in a data lake for off-line analytic use cases. Each of these delivery forms can have independent service level objectives, and the DataOps organization can track performance relative to those objectives when delivering data.

Tenet #2: Infrastructure as Code

A service can’t be highly available if responding to an issue in its infrastructure depends on having the person with the right knowledge or skills available. You can’t increase the capacity of a service if the configuration of its services isn’t captured anywhere other than in the currently running instances. And you can’t trust that infrastructure will be correctly deployed if it requires a human to correctly execute a long sequence of steps. By capturing all the steps to configure and deploy infrastructure as code, not only can infrastructure changes executed quickly and reliably by anyone on the team, but that code can be planned, tested, versioned, released, and otherwise take full advantage of the depth of experience we have with software development.

With infrastructure as code, deploying additional servers is a matter of running the appropriate code, dramatically reducing the time to deployment as well as the opportunity for human error. With proper versioning, if an issue is introduced in a new version of a deployment, the deployment can be rolled back to a previous version while...
the issue is identified and addressed. To further minimize issues found in production, infrastructure can be deployed in staging and UAT environments, with full confidence that re-deploying in production will not bring any surprises. Capturing all infrastructure as code enables operations to be predictable, reliable, and repeatable.

From the DataOps perspective, this means that everything involved in delivering data must be embodied in code. Of course this includes infrastructure such as hosts, networking and storage, but, importantly, this also covers everything to do with data storage and movement, from provisioning databases, to deploying ETL servers and data processing workflows, to setting up permissions, access control, and enforcement of data governance policy. Nothing can be done as a one-off; everything must be captured in code that is versioned, tested, and released. Only by rigorously following this policy will data operations be predictable, reliable, and repeatable.

**Tenet #3: Automate Everything**

Many of the techniques available for keeping services available will not work if they require a human in the loop. When there is a surge in demand, service availability will drop if deploying a new server requires a human to click a button. Deploying the latest software to production will take longer if a human needs to run the deployment script. Rather, all of these processes need to be automated. This pervasive automation unlocks the original goal of making working software highly available to users. With pervasive automation, new features are automatically tested both for correctness and acceptance; the test automation infrastructure is itself tested automatically; deployment of new features to production is automated; scalability and recovery of deployed services is automated (and tested, of course); and it is all monitored, every step of the way. This is what enables a small DevOps team to effectively manage large infrastructure, while still remaining responsive.

Automation is what enables schema changes to propagate quickly through the data ecosystem. It is what ensures that responses to compliance violations can be made in a timely, reliable and sustainable way. It is what ensures that data freshness guarantees can be upheld. And it is what enables users to provide feedback on how the data does or could better suit their needs, so that the process of rapid iteration can be supported. Automation is what enables a
small DataOps team to effectively keep data available to the teams, applications and services that depend on it.

**DataOps Practices**

The role of the operations team is to provide the applications, services, and other infrastructure used by the engineering teams to code, build, test, package, release, configure, deploy, monitor, govern, and gather feedback on their products and services. Thus, the operations team is necessarily interdisciplinary. Despite this breadth, there are concrete practices that apply across all these domains:

*Apply Agile Process*

Short time-to-delivery, responsiveness to change, and everything that comes with it, are mandatory for the DataOps team to effectively support any other agile team.

*Integrate With Your Customer*

The DataOps team has the advantage that the customers, the engineering teams they support, are in-house, and therefore readily available for daily interaction. Gather feedback at least daily. If it’s possible for DataOps and Data Engineering to be co-located, that’s even better.

*Implement Everything in Code*

This means host configuration, network configuration, automation, gathering and publishing test results, service installation and startup, error handling, etc. Everything needs to be code.

*Apply Software Engineering Best Practices*

The full value of infrastructure as code is attained when that code is developed using the decades of accumulated wisdom we have in software engineering. This means using version control with branching and merging, automated regression testing of everything, clear code design and factoring, clear comments, etc.

*Maintain Multiple Environments*

Keep development, acceptance testing and production environments separate. Never test in production, and never run production from development. Note that one of the production environments for DataOps is the development environment for the data engineers, and another is the production environment for the data engineers. The DataOps development environment
is for the DataOps team to develop new features and capabilities.

Integrate the Toolchains

The different domains of operations require different collections of tools (“toolchains”). These toolchains need to work together for the team to be able to be efficient. Your data movement engine and your version control need to work together. Your host configuration and your monitoring need to work together. You will be maintaining multiple environments, but within each environment, everything needs to work together.

Test Everything

Never deploy data if it hasn’t passed quality tests. Never deploy a service if it hasn’t passed regression tests. Automated testing is what allows you to make changes quickly, having confidence that problems will be found early, long before they get to production.

These practices enable a small operations team to integrate tightly with data engineering teams, so that they can work together to deliver the timely, high-quality data that powers applications and analytics.

DataOps Challenges

DataOps teams, particularly those working with big data, encounter some challenges that other ops teams do not.

Application Data Interface

When integrating software packages into a single product, software engineers take advantage of application programing interfaces (APIs), which specify a functional and nonfunctional contract. Software subsystems can be written to provide or consume an API, and can be independently verified using a stubbed implementation on the other side of the API. These independently developed subsystems can then be fit together, and will interoperate thanks to the contractual clarity of the API. There is no such equivalent for data. What we would like is an application data interface (ADI), which specifies a structural and semantic model of data, so that data providers and data consumers can be verified independently, then fit together and trusted to interoperate thanks to the contractual clarity.
of the ADI. There have been multiple attempts to standardize representation of data structure and semantics, but there is no widely accepted standard. In particular, the DDL subset of SQL specifies structure and constraints, but not semantics, of data. There are other standards for representing data semantics, but none has seen broad adoption. Therefore, each organization will need to independently select and employ tools to represent and check data model and semantics.

Data Processing Architecture

There are two fundamental modes for data: snapshots, represented in tables, and transactions, represented in streams. The two support different use cases, and, unfortunately, they differ in every respect, from structure, to semantics, to queries, to tools and infrastructure. Data consumers want both. There are well-established methods of modeling the two in the data warehousing world, but with the ascendency of data lakes we are having to discover new methods of supporting them. Fortunately, the data warehousing lessons and implementation patterns transfer relatively cleanly to the technologies and contexts of contemporary data lakes, but since there is not yet good built-in tool support, the DataOps team will be confronted with the challenge of assembling and configuring the various technologies to deliver data in these modes.

There are now multiple implementation patterns that purport to handle both snapshot and streaming use cases, while enabling a DataOps team to synchronize the two to a certain degree. Prominent examples are the Lambda Architecture\(^3\) and Kappa Architecture\(^4\). Vendor toolchains do not yet have first-class support for such implementation patterns, so it is the task of the DataOps team to determine which architecture will meet their organization’s needs, and to deploy and manage it.

Query Interface

Data is not usable without a query interface. A query interface is be a type of API, so data consumers can be written and verified against an abstract interface, then run against any provider of that API.

\(^3\) http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html
\(^4\) https://www.oreilly.com/ideas/questioning-the-lambda-architecture
Unfortunately, most query interfaces are vendor or vendor / version specific, and the vendors only provide one implementation of the query interface, so much of the benefit of writing to an API is lost. SQL is an attempt to create a standard data query API, but there is enough variation between vendor implementations that only the simplest of queries are compatible across vendors, and attaining good performance always requires use of vendor- specific language extensions.

Thus, even though we want to focus on data as a service independent of any particular vendor platform, the current reality is that the vendor and version of most query interfaces must be transparent to end users, and becomes part of the published interface of the data infrastructure. This impedes upgrades, and makes it nearly impossible to change vendors.

This problem is compounded by the fact that different data consumers require different kinds of query interface to meet their needs. There are three very different modes of interacting with data, and the DataOps team needs to provide interfaces for all of them:

1. A REST interface to find, fetch, and update individual or small groups of records
2. A batch query interface that supports aggregation over large collections of data
3. A streaming interface that supports real-time analytics and alerting

The infrastructure, technology, and design of systems to support each of these kinds of query interface is very different. Many vendors provide only one or two of them, and leave much of the complexity of deployment up to the DataOps team. The DataOps team needs to take this into consideration when designing their overall data processing architecture.

**Resource Intensive**

Even moderate scale data places significant demands on infrastructure, so provisioning is another DataOps challenge. DataOps needs to consider data storage, movement, query processing, provenance, and logging. Storage must be provisioned for multiple releases of data, as well as for different environments. Compute must be provi-
sioned intelligently, to keep data transfers within acceptable limits. Network must be provisioned to support the data transfers that cannot be avoided. Although provisioning to support resource-intensive loads is not unique to DataOps, the nature of data is such that DataOps teams will have very little runway relative to other kinds of teams before they start to run into difficult challenges and tradeoffs.

**Schema Change**

Vendors change data with every release. Analysts require data changes for every new analytic or visualization. These modifications put schemas, and therefore ADIs, in a state of perpetual change. Each change may require adjustment to the entire depth of the associated data pipelines and applications. Managing the entire DataOps ecosystem as versioned, tested code, with clear separation between development and production environments, makes it possible to respond quickly to these changes, with confidence that problems will be caught quickly. Unfortunately, many tools still assume that schemas change slowly or not at all, and the DataOps team must implement responsiveness to schema change outside these tools. Good factoring of code to centralize schema definition is the only way to keep up with this rapid pace of change.

**Governance**

Regulations from both government and industry cover data access, retention, traceability, accountability, etc. DataOps must support these regulations, and provide alerting, logging, provenance, etc. throughout the data processing infrastructure. Data governance tools are rapidly maturing, but interoperability between governance tools and other data infrastructure is still a significant challenge. The DataOps team will need to bridge the gaps between these toolchains, to provide the coverage required by regulation.

**The Agile Data Organization**

DataOps in conjunction with agile data engineering builds the next generation data engineering organization. The goal of DataOps is to extend Agile process through the operational aspects data delivery, so that the entire organization is focused on timely delivery of working data. Analytics is a major consumer of data, and DataOps in the
context of agile analytics has received quite a bit of attention. Other consumers also benefit substantially from DataOps, including governance, operations, security, etc. By combining the engineering skills that are able produce the data, with the operations skills that are able to make it available, this team is able to cost-effectively deliver timely, high-quality data that meets the ever-changing needs of the data-driven enterprise.

This cross-functional team will now be able to deliver several key capabilities to the enterprise:

**Source Data Inventory**
Data consumers need to know what raw material is available to work with. What are the data sets, and what attributes do they contain? On what schedule is the source updated? What governance policies are they subject to? Who is responsible for handling issues? All of these questions need to be answered by the source data inventory.

**Data Movement and Shaping**
Data needs to get from the sources into the enriched, cleaned forms that are appropriate for operations. This requires connectivity, movement, and transformation. All of these operations need to be logged, and the full provenance of the resulting data needs to be recorded.

**Logical Models of Unified Data**
Operations need to run on data models that are well-understood, of entities that are tied to the business. These models need to be concrete enough to enable practical use, while maintaining flexibility to accommodate the continuous change in the available and needed data.

**Unified Data Hub**
The hub is a central location where users can find, access, and curate data on key entities - suppliers, customers, products, etc. - that powers the entire organization. The hub provides access to the most complete, curated, and up-to-date information on these entities, and also surfaces the provenance, consumers, and owners of that information.

---

Feedback

At time-of-use, data quality issues become extremely transparent, so capturing feedback at point-of-use is critical to enabling the highest-quality data. Every data consumer needs a readily accessible feedback mechanism, powered by the Unified Data Hub. This will ensure that feedback can be incorporated reliably and in the most timely manner.

Combining DataOps with your agile data engineering organization will allow you to achieve the transformational analytic outcomes that are so often sought, but that so frequently stumble on outdated operational practices and processes. Quickly and reliably responding to the demands presented by the vast array of enterprise data sources and the vast array of consumption use cases will build your “company IQ.” DataOps is the transformational change data engineering teams have been waiting for to fulfill their aspirations of enabling their business to gain analytic advantage through the use of clean, complete, current data.
About the Authors

**Andy Palmer** is co-founder and CEO of Tamr, a data unification company, which he founded with fellow serial entrepreneur and 2014 Turing Award winner Michael Stonebraker, PhD, adjunct professor at MIT CSAIL; Ihab Ilyas, University of Waterloo; and others. Previously, Palmer was co-founder and founding CEO of Vertica Systems, a pioneering big data analytics company (acquired by HP). During his career as an entrepreneur, Palmer has served as founding investor, BOD member or advisor to more than 50 start-up companies in technology, healthcare and the life sciences. He also served as Global Head of Software and Data Engineering at Novartis Institutes for BioMedical Research (NIBR) and as a member of the start-up team and Chief Information and Administrative Officer at Infinity Pharmaceuticals. Additionally, he has held positions at Bowstreet, pcOrder.com, and Trilogy.

**Michael Stonebraker** is an adjunct professor at MIT CSAIL and a database pioneer who specializes in database management systems and data integration. He was awarded the 2014 A.M. Turing Award (known as the “Nobel Prize of computing”) by the Association for Computing Machinery for his “fundamental contributions to the concepts and practices underlying modern database systems as well as their practical application through nine start-up companies that he has founded.”

Professor Stonebraker has been a pioneer of database research and technology for more than 40 years, and is the author of scores of papers in this area. Before joining CSAIL in 2001, he was a professor of computer science at the University of California Berkeley for 29 years. While at Berkeley, he was the main architect of the INGRES relational DBMS; the object-relational DBMS POSTGRES; and the federated data system Mariposa. After joining MIT, he was the principal architect of C-Store (a column store commercialized by Vertica), H-Store, a main memory OLTP engine (commercialized by VoltDB), and SciDB (an array engine commercialized by Paradigm4). In addition, he has started three other companies in the big data space, including Tamr, oriented toward scalable data integration. He also co-founded the Intel Science and Technology Center for Big Data, based at MIT CSAIL.
**Nik Bates-Haus** is a technology leader with over two decades of experience building data engineering and machine learning technology for early stage companies. Currently, he is a technical lead at Tamr, a machine learning based data unification company, where he leads data engineering, machine learning and implementation efforts. Prior to Tamr, he was director of engineering and lead architect at Endeca, where he was instrumental in the development of the search pioneer which Oracle acquired for $1.1B. Previously, he delivered machine learning and data integration platforms with Torrent Systems, Thinking Machines Corp., and Philips Research North America. He has a master’s degree in computer science from Columbia University.