Chapter 1. Introduction

Data-Driven Approach

As software developers, we produce much of the intellectual capital that fuels economic and social change in today's digital world. The proliferation of data—and the rapid development of new tools to manage it—have put new demands on us. For 40 years, we put data into relational database management systems (RDBMSs)—we knew what the data looked like and database administrators (DBAs) knew what to do to store, secure, and retrieve that data. Now, with a variety of different data types and unprecedented data volumes, we are finding that as data becomes more complex and siloed in new purpose-built databases, we are being forced away from it. This creates ever-longer lags between specification and deployment.

Data-driven software development accepts the central role that data in its primary form takes in the applications that software developers create (Figure 1-1). Direct access to a multi-model database gives developers the ability to implement the transformations they need to accomplish. Documents are the heart and soul of information technology: two systems exchanging

Outside of IT, much of what people work with is document-oriented as well. Everyday transactions such as sending emails, viewing pages in a web browser, filling out forms, creating Word documents, and doing financial modeling in Excel are all document-centric activities that are not represented naturally in the restrictive relational data context. Multi-model databases accept data in the form the world already understands.

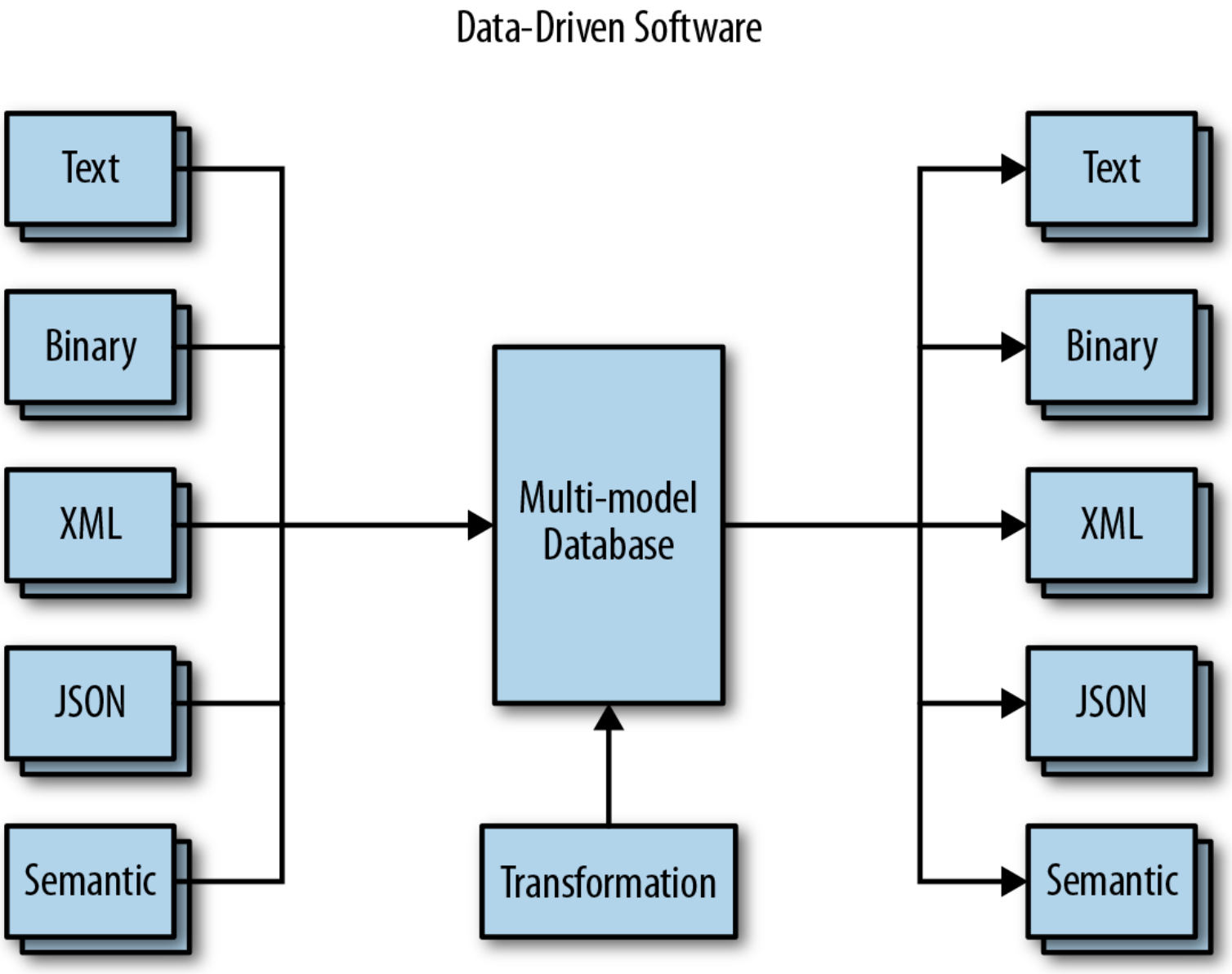

Figure 1-1. Data-driven software development

The data-driven approach requires a new way of [illegible]-ing into a multi-model environment allows us to rapidly develop, providing new-

Collection of Case Studies

As the following case studies and many other notable examples demonstrate, a data-driven approach and using a multi-model database enables business solutions that older approaches and older database technology simply could not accomplish within any tolerable time and budget constraints.

Fortune 50 Insurance Company Case Study

The first case study we'll look at involves a Fortune 50 insurance company that provides coverage for more than 20 million Americans. The company needed a better way to organize records for their roughly 25,000 employees.

Over the years, the insurance company had acquired a number of other firms, but its employee information remained in those firms' independent databases.

The company wanted to integrate its human resources data into a central repository so that it could be delivered to downstream systems for other real-time apps and batch analytics. The feeds included employee data, payroll data, employee evaluation data, and many other systems.

While much of the data was relational, integrating meant developing complex point-to-point data exchange solutions involving highly complex extract, transform, and load (ETL) processes that had to be written up front. This approach would be costly, time consuming, and perhaps impossible because of the rigid constraints of relational databases. It would also result in a brittle system that would require massive rework for any future changes.

Instead, the company created a flexible, document-based data model that could be developed in an Agile manner. This eliminated the need for a lot of complex ETL. The model was geared for communications between different formats. making it possible to decouple source data from target da ar making applications more resilient to change an ta

The HR Data Hub successfully integrated more than 24 different HR data sources comprising about 140 data feeds. Today the data is consumed by more than 50 systems in over 200 different formats. The HR Data Hub handles terabytes of data and has a throughput of about 50 GB of data per day. The company delivered the solution in less than a year—a notable improvement over the five years it had previously been estimated to take.

Fortune 50 Bank Case Study

We'll now consider the example of a Fortune 500 bank that needed a better way to organize tens of thousands of IT assets, including everything from data centers, racks and servers, network routers, and applications.

Understanding the relationships between these assets and which individuals or services are impacted if they fail is critical to maintaining the bank's operations.

The data describing these different assets originates in a wide variety of data sources, and the bank needed to be able to perform analysis such as:

- Identifying affected software systems and business units when issues arise
- Cost reporting at a software system level
- Impact analysis and reporting for data center and infrastructure changes
- Effective, Google-like search across multiple silos
- Efficient data entry with reduced inconsistencies across various compliance initiatives
- Identification of gaps and redundancies ac

imagine, the data becomes hard to query through SQL, as there is no common data model linking the more-than-100 data sources that need to be integrated.

Instead of using a relational approach, the bank built a data repository and brought the data in as documents enriched with semantic triples (to describe links among assets). The result of this project is a technical asset inventory system that integrates all data silos and provides a real-time search and query interface to better analyze status, cost, impact analysis, gaps and redundancies, and much more.

Using document search and semantic queries along with SQL queries allowed the bank to greatly speed up development and improve capabilities over what would have been possible with a relational approach.

Additionally, the bank added a temporal layer to analyze events that occurred in the past and determine the impact of changes in its system.

Special-Purpose Solutions

The examples just discussed involved large and otherwise unmanageable situations. Sometimes we can find a *point* solution—or a special-purpose solution (or product)—that solves a specific problem rather than addressing all of the requirements that might otherwise be met with a multipurpose or multiservice product. Point solutions individually and collectively represent tremendous opportunities for improving application performance and for simplifying individual tasks. However, there is a danger that in rushing to solve one problem, we might create others. Here are some examples:

Security audit reveals vulnerabilities across multiple tools storing data in multiple formats.	Work to implement multiple security measures across multiple tools and across multiple versions on an ongoing basis.	Full acceptance of the security demands of your application could have led to the selection of a multi-model database that directly addressed all the security requirements.
Queries are not effective because they are hard to compose, expensive to execute, and/or take too long to complete.	Add an analytical database and/or add a memory-based database.	Duplicating operational data can introduce inaccuracy and timeliness challenges.
Rigid schemas and "master data models" limit rate of change for large projects.	Add a new NoSQL database to allow storage of heterogeneous records.	Many features of traditional relational systems can be lost (e.g., ACID transaction support and record validation against schemas).
Semantic data does not fit well into traditional datastores.	Add a new triple datastore specialized for semantic data.	The triple datastore requires separate management, and synchronization can be complex. Also, the triple store will require its own security plan, implemen-

Full-text search is required across both documents and operational records.	Add a full-text search engine and incorporate indexing with all record and document transactions.	Search makes all create, update, and delete (CRUD) operations more complex, while the accuracy of search results remains dependent on accurate synchronization. Also, the search tool will require its own security plan, implementation, maintenance, and audit.

Developing a Broader Perspective on Data Management

Developers can add value to their companies by focusing on delivering solutions faster and more securely than ever with a perspective that includes security, analysis, and operations concerns. Doing so will help them responsibly advance better approaches to data management.

As developers, we have seen this to some degree with the advancement of development operations (DevOps). DevOps is a collision of two disciplines of operations and development engineering that forged a new way of collaborating and problem solving. DevOps encouraged both camps to embrace some of the challenges previously encapsulated inside operations. In return for our investment in that, we have inherited many new processes and tools.

In a way, DevOps offers us a chance to frame our collective problems as collective opportunities:

"When your Ops folks start thinking about de[illegible] starting thinking about operations, you have a single tea[illegible] uous development, continuous testing, and continuous deployments."

mon problems that could benefit from a new approach:

A query across operational data cannot be run during the day because it would overload the servers.	With a multi-model database, specialized indexes combining full-text and traditional SQL concepts of entities can be built directly into the database, where indexes are constantly and consistently maintained. This enables fast and accurate search without extraordinary system loading.
The schema is not set up to do that.	With a multi-model database, multiple schemas can be used within the same collection. Collections with no schemas (NoSQL) are also possible.
We have no approved tools to store triples.	With a multi-model database, triples can be stored alongside the documents and records to which they refer.
We cannot do a keyword search on that database.	With a multi-model database, all parts of all documents and records can be referred to through a single, integrated index.
We sharded the NoSQL database last year, and we cannot change the key now.	With a multi-model database, there is no need to select a single shard key early in an application lifecycle.

The NoSQL database is not in our disaster recovery plan, so we lost it when we restored the system.

With a multi-model database, structured and unstructured data is maintained with a single set of high-availability (HA) and disaster recovery (DR) mechanisms.

With a clearer idea of the dramatic impact that our data management tools can have on the success or failure of our projects, let's take a closer look at what this data is in the real world.

---

[1] New Relic website

Teach, write, train
Careers
Community partners
Affiliate program
Submit an RFP
Diversity
O'Reilly for marketers

SUPPORT

Contact us
Newsletters
Privacy policy

your phone and tablet.

Download on the App Store GET IT ON Google Play

WATCH ON YOUR BIG SCREEN

View all O'Reilly videos, Superstream events, and Meet the Expert sessions on your home TV.

Roku Players & TVs available at amazon appstore

DO NOT SELL MY PERSONAL INFORMATION