MARTIN DILGER

Understanding Eventsourcing

Planning and Implementing scalable Systems with Eventmodeling, Eventsourcing and Axon

First edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

1

Table of Contents

Part II - Modelling the System

Brainstorming
Modeling Use Cases with Wireframes
"Given / When / Then" Scenarios
Use Case: Clear Cart
Use Case: Submit Cart
Use Case: Inventory Changed
Use Case: Price Changed
Structuring Eventmodels

Part III - From Zero to Running Software

Technology Stack
Brief introduction to Axon
Implementing the first slice - "Add Item"
Implementing State Views Slices using Live-Projections
Implementing Remove-Item and Clear-Cart
Example Integration with Apache Kafka and Translations
Implementing a database projection for inventories
Implementing Automations
Submitting the Cart
Handling breaking changes

Part IV - Implementation Patterns

What this part is about
Pattern: Database Projected Read Model
Pattern: Live Model Pattern: The (partially) synchronous

2

Why I care

Initially this chapter wasn´t planned. But many people asked, why I actually cared. Why do I make this effort. What drives me.

I began developing software around 2005, so I'm nearing my 20th anniversary in this business. It's kind of incredible, as I don't feel like it's been 20 years at all. But one thing has not changed: Software Development seemed hard back then, and it seems hard today.

However, I have also seen different projects. Projects that did well. That worked, were carefully planned and successful. I think overall we can do better, if we learn to ignore all the distractions that divert us from what we should truly be focusing on—delivering value.

I once had a client say, "Software developers are like pizza deliveries: they're always late, often deliver only half of what they promised, and it's never as good as it looks on the shiny prospect."

In essence, he's right. In my opinion, software development hasn't changed much in the last 20 years—maybe not even in the last 40 years. Agile, microservices, cloud, and AI haven't fundamentally altered that fact. Software development is still like the pizza nobody wants in the end. But you have to eat it because you ordered it.

However, in 2021, I learned about Event Modeling as something that was about to change everything for me. I finally could talk to my customers in a language they understood. I could discuss the problems that truly bother them without confusing them with technical terms and details they neither understood nor cared about.

We could suddenly focus entirely on one thing: the real problem to solve.

We found that the root of many issues wasn't premature optimization, as many developers think. It was the lack of a common language—the lack of a structured way to communicate.succesful

This is not a new thing. The Book "Domain Driven Design" by Eric Evans talks about this common language between IT and Business at length. He famously named it the "Ubiquitous Language". We will discuss this in more detail in chapter 8.

Alberto Brandolini, the inventor of Evenstorming[1] famously said, "It's developers understanding, not your business knowl-

[1] https://www.eventstorming.com/

edge that becomes software."[2]

But what if the developer simply gets it wrong? What if the lack of a common language, the absence of clarity in terminology used, and the lack of time to properly discuss the problems to solve result in constant misunderstandings between business and development? What if this is the problem we've been trying to tackle for 20 years with one technical solution after another?

Event Modeling seemed magical to me back then. Every time I used it, it worked, even with people I had never met before. Suddenly, everyone understood. Suddenly, we could talk to each other. I began using it whenever I could. At first, I was unsure if I was doing it correctly, but more and more, I felt confident because it was actually simple—almost too easy.

Finally, there was clear communication. We could confidently discuss the business and the problems we aimed to solve next. More importantly, we could talk about the cost of delivering valuable solutions that would address these problems once and for all. We gained a lot of trust in the systems we planned and got rid of that lingering uncertainty—the nagging feeling that something important was being overlooked.

I've seen really bad things: huge amounts of money burned for nothing, waste of time, labor, and knowledge. Meetings with endless discussions but no outcomes. Requirements that nobody truly understood but were used to estimate and plan for months and years ahead - for a product that in the end took

[2] https://x.com/ziobrando/status/1347126001340358656

much longer than initially planned.

For me I decided from then on, that I wanted to pursue a different way. I decided that our industry can do better. And I realized, that we don't have to change everything to get there. It´s just some minor tweaks. In January 2024, I said in a webinar [3], Software Development will fundamentally change within the next 2 years. And I think we are on the way as an industry.

This book is about everything I learned in the last 15 years. It is about what I discovered in 2021. Tiny changes in the process, focus on the real problems, communication in a language that is understood by everyone makes all the difference you need.

Internally, we have a name for it. We call it "accelerate". Teams working with these ideas should feel "accelerated" pretty quickly. The longer you work with this approach, the faster you iterate. Software development can be a well-oiled machine, constantly going from planning to development and back. Embracing change as part of the game, without slowing down. You will find a description of this development process and how I typically work in the Appendix about "Accelerate".

I´m standing on the shoulder of giants. None of the ideas presented in this book are my very own. I just combine what some people way smarter than me came up with like Adam Dymitruk, Greg Young, Jimmy Bogard, Eric Evans and Vaughn Vernon.

---

[3] https://nebulit.de/bessere-software-2024

This is what drives me and I can´t wait. I want to see what will change when we start to do things this ways.

Martin, July 2024

PS: One more thing maybe.. I put my heart into this book and with the help of many, many people managed to write it in 3 months. I hope it´s a great book, but it´s certainly not perfect. Not yet.

If you find issues, help me to make it better. Open issues here on github:

https://github.com/dilgerma/eventsourcing-book/issues

Let´s make this the best book it can be.

3

Why you should care

What is Event Sourcing and why should you care as a developer, team manager, or company? Event Sourcing is far from a new technology; it emerged around 2006 in the software industry. So it's about 15 years at the time of writing this book since it hit the software development mainstream (and much older if you trace it back to its ancient origins).

So why would you care?

Event Sourcing as an implementation pattern is rather simple. Instead of storing your data in tables and relations, you store the facts that actually happened in the system in the order they happened.

This means the data in your system is like a simple story of what happened recently.

First, the customer was created, then the customer moved to a

new address. Then the customer made a big purchase. Because of that, the customer got promoted to premium status, and so on.

If you add up all these facts (in Event Sourcing we call the facts "events"), you get a customer with a new address in premium status.

Although you get the same customer with the traditional CRUD-based approach, where a system stores data in a normalized relational schema, what you lose is the history of how the customer became a premium customer over time.

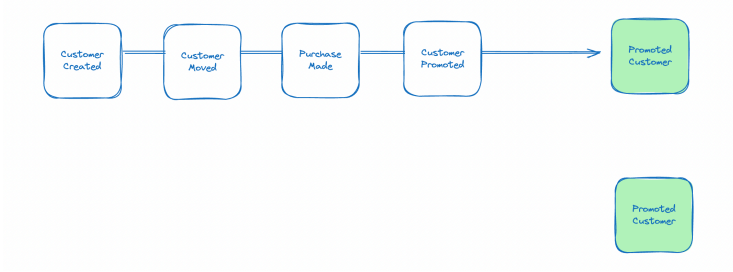

Fig 1.1 - Historical Data (top) / flattened data (bottom)

So in the end, you have the same result. Why would it matter in the first place?

It starts to matter when your company wants to work with the

data. What if it was important to know if the customer became premium before or after a certain point in time?

What if you wanted to know how much money people spend on new furniture within two weeks after the customer has moved to a new address?

What if you needed to know when the customer's address was changed and how often?

By using Event Sourcing, we are adding the dimension of time to our system. The system records the facts that happen along a timeline. This dimension gets lost when data is flattened to a relational database schema.

Of course, you can consult your data warehouse (if you have one) to get back some of the data. In simple terms, a data warehouse is a big database to capture all relevant data changes over time. But what would happen if all this data would be freely available and could be used for whatever ideas your business unit comes up with in the future?

What if you could feed an AI model with the data you collected in the last 10 years? And what if this model would help you explore new markets based on hard facts rather than blank theories?

The only way for your company to truly learn and benefit from the past is keep detailed records. This all requires some new thinking and a different approach. And it requires the information to be accessible in a flexible format.

Using Event Sourcing makes it easier to reason about your system without quickly falling into the trap of thinking in purely technical terms. It's much easier to understand a system if it's modeled the way it really works.

Instead of saying, "We need to set this blocked flag on the customer" (which no business person would ever understand), it's much more natural to create a new event named "Customer Blocked." This event indicates that a customer was blocked at a certain point in time and for a specific reason.

But unfortunately, this way of thinking is rarely taught in schools or universities, and that's why many developers struggle with this simple concept even after years in the software industry.

"Not losing information" is the foundation of Event Sourcing. We keep information readily available at all times because you never know what it might be useful for in a few years.

Learning Event Sourcing is similar to learning a new language. You often need to search for how things are done using this approach. However, if you try googling or asking ChatGPT, you'll quickly realize that finding information about these concepts isn't as straightforward as you would expect. Although the information is out there, it's scattered across many different sources. Plus, there are significantly fewer books on Event Sourcing compared to those covering traditional information systems.

So now, again the question: why would you care?

Because information is the new gold. Data helps you make informed business decisions. And keeping data available comes basically as a by-product when working with Event Sourcing.

In this book, we will dive deeper into Event Sourcing and everything around it, mainly focusing on the implementation side of it. That is what we lack most - clear guidance of how to actually do it using real world examples.

Of course, we will discuss the theories around it and what it means to work like that. But compared to other books and information available, we will try to dive much deeper into the real implementation of it to really understand what it means to build an event-sourced system.

But not only that.

We'll also delve deeper into the planning phase of event-sourced systems, using collaborative modeling techniques you might or might not be familiar with. Our main focus will be on Event Modeling, a technique developed and refined by Adam Dymitruk over the past decade[4]. Event Modeling helps us plan and discuss information systems in a clear language. While it's not limited to event-sourced systems, you'll find that it's a particularly good fit for them.

So what will you learn in this book?

You will learn pretty quickly what took me 15 years to learn.

---

4 https://eventmodeling.org/

It's entirely possible to build plannable, scalable, and well-structured information systems consistently, not just once, with the right approach and a solid plan. In this book, I'll share how I create these plans. Some of the content reflects my own perspectives, and you may disagree with certain points—that's understandable. This book is not abstract; it dives deeply into the details of Event Modeling, Event Sourcing, and the implementation aspects behind them. I recognize that some readers might find this challenging, and I apologize if that's the case.

We'll dive into Event Sourcing patterns, where you'll learn some techniques that aren't secret but aren't widely shared either, which can help you implement an event-sourced system in a scalable way.

I'll share how, after 15 years of developing information systems, I've structured my architectures and systems in a modular way. This may not be the 'best' or 'perfect' approach, but it's the one that has worked well for me.

I'll keep things as brief and focused as possible, leaving out anything that isn't crucial to your success.

Believe me, implementing event-sourced systems is simpler than you might expect—as long as you have a clear roadmap and someone to guide you. I wish I'd had a resource like this when I started, so my goal is to help you along your journey and get you started today.