Member-only story

Understanding and Solving the Primary Data Analysis Problem

Why data analytics fails before any analysis begins

9 min read · Just now

Gunnar Östberg

In this story, I describe a fundamental issue in data analysis that business and IT professionals struggle to understand. I try to describe this problem and its solutions in a way everyone can absorb.

Introduction

The large, standard systems that are popular today and commercially successful were originally designed some 20 years ago and have since been expanded. I am referring to ERP, CRM, trading, and contract management systems for finance and insurance, as well as for other industries. They may have been built initially for one specific customer and then delivered to tens, hundreds, or thousands of other customers, and expanded significantly, in many small steps, usually without changing the information or database architecture. These information systems were designed long before data analysis was used as a business process. They now comprise a mix of core systems of varying ages and technologies in a company's IT environment. Their databases are a source for data analysis and sometimes feed data warehouses, data lakes, or similar.

Many data analytics initiatives fail because business meaning is lost long before data reaches analysts.

Business concepts and meaning

In the needs analysis prior to introducing the standard system in the business, a *concept model* was typically developed as one of the artefacts. This model of business concepts is necessary so that we know what we are talking about and agree on what we mean. The concept model defines concepts, describes terms, and specifies relationships and frameworks. Each concept is described with a name (how do we denote this concept), a text (how do we describe this concept), perhaps a few attributes (how do we characterize this concept), and relationships to other concepts. A concept model typically contains hundreds of business concepts. The concept model is very important because all human-made business decisions are based on a common understanding of the business.

At this level, the concept model's purpose is purely semantic. It exists to align human understanding, not to design systems. A concept model answers questions such as: What is a customer? What is a contract? How do these concepts relate? What are they not? The model deliberately avoids technical details, identifiers, storage concerns, and system constraints. Its job is to establish a stable business language that survives organizational change, system replacements, and product evolution.

To illustrate this, the following concept model examples are drawn from the life and pension insurance domain, where the concepts of customer and contract are central but frequently misunderstood and overloaded in practice. Please note that I use the life and pension insurance domain as an example; the fundamental issue this story handles applies to all industries and domains. Please also note that the examples are much simplified.

Example: Concept model: Customer

Let's start with "customer", and some examples.

> *What do we mean when we say "customer"?*

The "customer" overload happens in four ways.

1. **Example: Concept model: Customer: Same term, different concepts**
   In a single policy administration system, "customer ID" might reference a master party record containing KYC data (e.g., name/address) across all policies for reporting queries, but reference the policyholder role and contract-specific attributes (e.g., premium payment method) in billing runs.
   The same API endpoint `/customer/{id}/policies` uses "customer" for the party in some insurers' docs, but for the primary policyholder in others. The same term is used for different concepts, and customer ID is a term frequently used in the business of a life and pension insurance company.

2. **Example: Concept model: Customer: Different terms, same concept**
   The underlying legal owner of the policy, who signs the contract, pays premiums, controls changes, is called the policyholder in core admin processes and systems, the policy owner in sales portals, the contract owner in actuarial models, and simply the customer in CRM processes and tools.

3. **Example: Concept model: Customer: Different stakeholders or domains**
   - Marketing treats "customer" as any prospect who inquired about a pension plan (lead data).
   - Claims uses "customer" to refer to the beneficiary submitting a death claim.
   - Distribution partners for group pensions call the employer "the customer", while servicing partners call individual members "customers".

- The staff at the information system supplier works daily with all the above terms and concepts of customer, and uses “customer” for their client companies and client actors as well.

4. **Example: Concept model: Customer: Umbrella concept with multiple roles**
   The same natural person can appear as “customer” wearing different roles on one contract: Policyholder (owns/pays), Insured Person (whose life is covered), Beneficiary (receives payout), Payer (if premium auto-debit differs), and even Employer in group pension wrappers where they sponsor for employees. Roles are distinct concepts and not subclasses of “customer”. All these terms, including customer, are used in the life and pension insurance business. The concept model for “customer” may look like:

- Customer → *may act as* → Policyholder
- Customer → *may act as* → Insured Person
- Customer → *may act as* → Beneficiary
- Customer → *may act as* → Payer
- Customer → *may act as* → Employer

Example: Concept model: Contract

As a second example, we have “contract”. Like “customer”, “contract” is a widely used conceptual term used daily in the business of a life and pension insurance company. However, there is no single “contract”.

The basic question here is

> *What kinds of agreements actually exist in the business?*

Legal theory vs business usage

Theoretically, purely abstract, a contract is a concretized form of a valid agreement. An insurance contract represents a formalized, documented, and stored version of a valid agreement, in which the abstract mutual assent gains specific enforceability through written terms, signatures, and regulatory compliance.

In the business, though, a contract can have pre- and post-valid states. It is also used for entirely different information objects.

1. **Example: Concept model: Contract: Different agreements**
   We have:

- Insurance Policy
- Savings Agreement
- Payout Agreement
- Investment Mandate
- Management Agreement
- Employer Scheme Agreement

where the Insurance Policy refers to the other agreements. Each of these is a distinct legal and business concept with different rules, lifecycles, and relevance to analytics and decision-making.

Non-identical concept models

Each company, in other words, the system supplier's other customers, uses a slightly different set of concepts. The concept models of life and pension insurance companies are **not** identical, even though they might use the same information system. The concept models, part of the Enterprise Architecture, are managed and modified over time. Besides, differences in language, national culture, laws, and regulations give rise to distinct concept models. The same insurance company has variants of its concept model across its business operations in different countries.

Information requirements

In the requirements phase of implementing a core information system, the concept model is typically refined into an *information model* that specifies the information business systems need to handle to support the business's processes, decisions, and

regulatory obligations, independent of how or where that information will be stored.

At this level, business concepts are translated into information objects with clearly defined attributes, relationships, and constraints. For example, a customer is no longer treated as a single object but as a party that can assume different roles under different contracts over time. These roles become explicit information objects because they are operationally relevant and analytically significant.

The following information model examples show how the same business concepts are decomposed into information structures that support operational use.

Example: Information model: Customer roles

> *What information must the system handle to support the business?*

We have the following information objects tied to contracts and time:

- Party
  (Person or Organization)
- CustomerRole
  (roleType, validFrom, validTo)
- Contract

where the Party holds a CustomerRole, which relates to Contract. The roles are time-bound and contract-specific, and the same party can have multiple roles simultaneously.

Example: Information model: Agreement structures

> *What information is required to put requirements about contracts correctly?*

We have the following differentiation needed for correct analytics:

- Agreement
  (agreementId, agreementType, validFrom, validTo)

- Policy
- PaymentPlan
- BenefitPlan
- InvestmentAllocation

where the Agreement specializes in Policy/Payout/Investment and governs PaymentPlan, BenefitPlan, and InvestmentAllocation. The attribute or column agreementType drives structure and rules. Lifecycles are clearly expressed.

System design and implementation

In the design phase, the information model is normally broken down into a *data model* and a database model, which define how the information will be technically represented, optimized, and stored in specific information systems. The data model is rarely directly visible to customers and users, who instead use graphical interfaces or APIs. Except for application consultants and data analysts.

At this stage, practical concerns dominate. Performance, storage efficiency, legacy constraints, and system integration often drive design decisions. As a result, multiple business concepts are frequently flattened into generic tables such as CUSTOMER or CONTRACT, with meaning encoded implicitly through codes, flags, or naming conventions rather than explicit structure. This is the technical reality.

And that's how the system was built.

Example: Data/database model: Customer

See the designed and implemented tables:

- CUSTOMER
- CONTRACT
- CONTRACT_PARTY, with the columns:

- contract_id
- customer_id
- role_code

where the role_code is shown as opaque. e.g., ‘01’, ‘A’, ‘X’, and there are no validity dates (or shown as optional/missing), and there is no distinction between person and organization. The role's meaning is implicit and system-specific.

Example: Data/database model: Contract

The designed and implemented tables for the contract (legacy reality):

- CONTRACT
- CONTRACT_VERSION
- CONTRACT_RELATION

where CONTRACT has the following columns:

- contract_id
- parent_contract_id
- contract_type

in which the parent-child hierarchy is overloaded, and the contract_type is reused across meanings. The same table is used for:

- policy
- payout
- investment
- administration

Different business concepts share the same technical structure.

These last examples expose the root of the analytics problem. Aggregations and KPIs mix incomparable things.

The data analysis problem

To perform data analysis, you start with the data and analyze it to make decisions at the business and operational levels.

This is where it all goes wrong.

In the examples above, the original business meaning of customer and contract has been progressively diluted. Roles, lifecycles, and legal distinctions that were clearly expressed at the conceptual and information levels are no longer directly visible in the data. What remains are generic tables and cryptic codes that require prior system-specific knowledge to interpret correctly.

In such a data structure, it is nearly impossible to automatically infer what the data represents from a business perspective. A data analyst cannot reliably determine whether a customer record represents an insured person, a payer, or a beneficiary. Likewise, a contract record may represent a risk policy, a payout agreement, or an investment mandate. As a result, analytical logic becomes fragile, manual, and error-prone.

Remember that, despite their different business concepts and terms, all businesses may share the same system data structure. This is critical.

> The data model is not wrong. It is insufficient for analytics.

The information about the operational concepts has been lost, and the data cannot be properly analyzed. The business decisions risk becoming wrong.

This breakdown typically occurs at the “understand” and “trust” stages of the information chain, which I have described in more detail elsewhere on how data analytics supports business decision-making using the information chain model.

For data analytics to create value, the full information chain must be intact. Data must exist, be accessible, be understood in its correct business context, and be trusted. If the semantic link between data and business concepts is broken, no amount of technical sophistication in data engineering or analytics can compensate.

The underlying business problem

There are a few recurring reasons:

- Concept models are treated as disposable project artefacts rather than long-lived assets.
- Information models are scoped to individual systems or products, not to enterprise-wide meaning.
- Data models are optimized for delivery speed, performance, and legacy compatibility, not semantic clarity.
- Over time, business meaning migrates from structure into documentation, code, and people's heads.
- When data analytics is introduced later, it is forced to work bottom-up from whatever data happens to exist.

The problem is structural, not accidental. By the time analytics arrives, the business language has already been lost.

The business solution

To perform data analysis correctly, the available data (e.g., database table and column names) must be translated and transformed into the business's decision-making concepts and terms, and the data processing must be adapted accordingly. This is not a pure data engineering task; it requires deep business management competence and, at times, subject-matter expertise.

When designing new systems, the requirements analysis must account for data analysis as a core business process, not as an afterthought. This means preserving

business meaning across conceptual, informational, and technical layers.

Maintain business model artefacts, and treat them as your most valuable assets. This is why Enterprise Architecture exists.

If business meaning is not preserved across models and systems, no analytics initiative can succeed.

Data Analysis | Data Analytics | Enterprise Architecture | Business Development | System Design Concepts

Edit profile

Written by Gunnar Östberg

196 followers · 140 following

CEO Business Clarity. I write enjoyable, rewarding articles that retain their value. Please check my profile to see the various story topics!

No responses yet

Gunnar Östberg

What are your thoughts?