# Data Flow (Naive Approach for Data Flow Modeling)

That was our comprehension at the time, we thought it is related to Flink as a tool and we wrote this way, it was a naïve attempt to represent our Data flow and It helped us look for data engineering/design patterns and so it was the start of the data flow we presented using the different patterns we saw fit the most

**Step 1**: Use Map to enrich Data that we generated.

**Step 2**: Filter (Valid/Invalid Responses).

**Step 3**: Split (Promoters, Passives, Detractors).

**Step 4**: Key By (Group streams by questions that are part of a particular survey).

**Step 5**: Counters and aggregators (Use 3 Counters for each stream) / Calculate total count for each category.

**Step 6**: Map (Calculate NPS for each question and survey) based on the counters.

**Step 7**: Reduce (Merge Streams) : union or connect.

**Step 8**: Key By again (Survey/Questions).

**Step 9**: Aggregator (calculate overall NPS and Metrics).

**Step 10**: Write the final output to sink (Grafana/Prometheus).

# Domain Driven Design (DDD):

#### What is a Domain?

The word **Domain** used in the context of software engineering refers to the business on which the application is intended to be built.

What is a **Domain Logic**?

In the process of **application development**, domain logic or business logic is commonly used. Basically, **business logic** is an area of knowledge around which **application logic** revolves.

The **business logic** of an application is a set of rules and guidelines that explain how business objects should interact with each other to process modeled data.

**Domain-Driven Design:**

Suppose we have designed software using all the latest tech stack and infrastructure and our software design architecture is fabulous, but when we release this software in market, it

is ultimately the end user who decides whether our system is great or not. Also, if system does not solve business need is of no use to anyone; No matter how pretty it looks or how well architecture its infrastructure. According to **Eric Evans**, when we are developing software, our focus should not be primarily on **technology**, rather it should be primarily on **business**. Remember,

*“It is not the customer’s job to know what they want” – Steve Jobs Strategic and Technical design tools:*

**Domain-driven design** talks about two kinds of design tools, first one is **Strategic design**

**tools** and another one is **Tactical design tools**.

deal with tactical design tools but if we have knowledge and a good understanding of strategic design tools then it will help us in architecting good software.

And that’s why we’re considering a Domain-Driven design approach.

#### Advantages of Domain-Driven Design:

* It improves our craft.
* It provides flexibility.
* It prefers domains over interface.
* It reduces communication gap between teams through Ubiquitous Language

**Pattern**: Domain-Driven Design (DDD):

**Description**: a **software** **development** approach that focuses on understanding the **business** **domain**, its **needs**, and **complexities**, and then modeling the software system to match and solve **real-world** **problems** and scenarios in that domain.

**Purpose**: Be able to create a structured, and scalable data flow that aligns with our business goals.

**Output**:

A clear separation of concerns, with each subprocess handling specific aspects of the data flow.

Boundaries (Bounded Context) between subprocesses:

#### Preprocess: Activities before implementing NPS metrics, like setting up surveys or gathering initial customer data.

* + - **Core Process:** Implementing and calculating NPS metrics directly handling the core business logic around customer feedback and loyalty assessment.

#### Postprocess: The phase of evaluating and applying the insights gained from NPS metrics to drive improvements and monitor ongoing customer satisfaction.

**Patterns:**1) **Single Source of Truth (SSoT**)

**Description:** Maintain data consistency by using one authoritative source for all data references.

2) **No Lookup**:

**Description**: minimizes or eliminates queries to external sources for data.

### Output:

### Step 1:

Reads and imports data from upstream.

### Step 2:

Check and make sure data are complete and consistent (rich). Hence, standardized data is ready for further processing.

### Step 3:

While parsing data we made sure it is rich enough to avoid fetching additional data from an external source and adding more complexity

**Step 4:**

We ensure consistency with the Kafka source/ Processed and aggregated data sent to a Kafka cluster for storage (Output to Data Sink)

**Patterns:**

1. **Canonical Data Model**:

**Description**: Unify how data is represented across different systems.

1. **Data Translation**:

**Description**: Converting data from one format to another to ensure compatibility in different contexts

### Output:

**Step 1**:

### Ensures that incoming data is in the correct format and adheres to the defined CDM structure.

**Step 2**:

### Focuses on catching and handling any format errors early in the process.

### Pattern:Early Exit

### Description:A strategy where a process stops as soon as a condition is met, improving efficiency by reducing unnecessary work

### Output:

**Step 1**: ignoring non-conforming messages.

### Filtering irrelevant data that doesn't conform to the required format or lacks essential fields so we can ignore it early.

**Step 2:**

Ensures only relevant data (NPS questions) proceed to core processing. We filter out non-NPS questions.

This gives us a Validated data set for NPS processing

### Pattern: Splitter

### Description:Splits incoming data streams into multiple streams, useful for segregating NPS questions into valid and invalid responses in your system.

### Output:

**Step 1:**

Segregates NPS questions into valid and invalid answers.

**Step 2:**

Addresses and resolves any issues identified in invalid NPS answers.

**Step 3:**

Assigning errors to invalid answers.

**Step 4:**   
  
Each stream is categorized into promoters, detractors, and passives, based on our predefined criteria.

### Pattern: Aggregator

### Description:Combines data from multiple streams, which can be used to aggregate responses into categories like promoters, detractors, and passives. (Source: <https://www.enterpriseintegrationpatterns.com/> )

### *Output:*

**Step 1:**

### Setting counters for Valid Answers.

**Step 2:**

### Setting counters for Invalid Answers.

**Step 3:**

Calculating the count of each category that fits into NPS-relevant groups (promoters, detractors, passives)

**Step 4:**

We will collect and compile the data into a suitable format for our visualization tool.

### Pattern:Correlation

### Description: Links related elements across different data sets, such as associating responses with their respective questions and participants in NPS analysis.

### *Output:*

**Step 1:**

Organize valid responses according to the specific survey question they address.

**Step 2:**

Create separate streams or groups for each question, with the corresponding responses grouped.

**Patterns**:

1. Enricher:

**Description**: Adds additional information to data, potentially useful for enhancing data before visualization in tools

1. Data Transformation:

**Description**: Modifies data structure or format, for standardizing and preparing data for analysis, visualization, and storage later.

### *Output:*

**Step 1:**

We’ll Make sure our data is in the correct format for its end use, like standardizing tex