

Basándome en el libro de Piethein Strengholt, aquí tienes un resumen claro y organizado de las **Mejores Prácticas para Construir Soluciones de Consumo (DDSs)**.

El autor deja claro que es imposible tener una solución única para todos los casos de uso, ya que el lado del consumidor es muy diverso. Sin embargo, hay un conjunto de mejores prácticas que se pueden (y deben) aplicar siempre. Las agrupa en cuatro grandes áreas:

1. Requisitos de Negocio (El Punto de Partida Obligatorio)

La regla de oro: empezar siempre por el problema de negocio.

- **Tener un Plan a Largo Plazo:** No resuelvas problemas *ad hoc* con soluciones de un solo uso. Piensa en el valor a largo plazo que el caso de uso aportará al negocio. ¿Es un proyecto para aumentar ingresos, reducir costos, gestionar riesgos o mejorar la satisfacción del cliente?
- **Empezar con Brainstorming y un *Business Model Canvas*:** Reúnete con los equipos de negocio para entender qué valor quieren desbloquear, qué herramientas necesitan y cuáles son los criterios de éxito.
- **Definir Objetivos Claros:** Antes de escribir una línea de código, asegúrate de que los objetivos están bien definidos, detallados y completos. Esto incluye saber qué fuentes de datos se necesitan, si la solución debe operar en tiempo real o por lotes, y si el resultado será reutilizado por otros dominios.

2. Audiencia y Modelo Operativo (Las Personas)

El éxito depende de tener a las personas correctas con responsabilidades claras.

- **Definir los Roles:** Es fundamental definir quién hace qué dentro del equipo de dominio que construye el DDS. Los roles típicos son:
 - **Domain Owner:** El líder del negocio, responsable del “qué” y el “porqué”.
 - **Application Owner:** El líder técnico, responsable del “cómo”.
 - **Analysts / SMEs:** Expertos en la materia que definen los requisitos y entienden el contexto.

- **Data Engineers:** Construyen y mantienen los *pipelines* de datos.
- **Data Scientists:** Construyen los modelos predictivos.
- **Report Builder:** Crean los *dashboards* para los usuarios finales.
- **Equipos Pequeños y Multifuncionales:** La estructura ideal es un equipo pequeño (tipo DevOps, de 5 a 10 personas) que contenga todos estos roles y que se enfoque en un objetivo de negocio bien definido. Esto permite escalar, ya que múltiples equipos pueden trabajar en paralelo en diferentes problemas.

3. Requisitos No Funcionales (La Tecnología)

La pregunta central: ¿qué base de datos o tecnología de almacenamiento usar?

- **No Hay una “Bala de Plata”:** No existe una base de datos que sirva para todo. La elección es un **trade-off** que depende de muchos factores.
- **Hacerse las Preguntas Correctas:** Antes de elegir, considera:
 - **Estructura de los datos:** ¿Son estructurados, semi-estructurados?
 - **Predictibilidad de las consultas:** ¿Sabes de antemano qué preguntas se harán? Si es así, puedes pre-optimizar los datos.
 - **Tipo de operaciones:** ¿Necesitas **joins** complejos (donde las bases de datos relacionales son mejores) o lecturas simples y rápidas?
 - **Velocidad y volumen:** ¿Los datos llegan en tiempo real o por lotes? ¿Qué volumen de datos manejarás?
 - **Flexibilidad del esquema:** ¿Necesitas poder cambiar la estructura de los datos fácilmente (donde las bases de datos NoSQL son mejores)?
 - **Escalabilidad:** ¿Necesitas escalar horizontalmente?
- **Estandarizar Limitando las Opciones:** No quieres que tu empresa termine usando 50 bases de datos diferentes. La mejor práctica es definir un **conjunto reutilizable de tecnologías y patrones** (los *blueprints*) que cubran la mayoría

de los casos de uso (ej: un estándar para BI, otro para *machine learning*). ¡La palabra clave del autor es “estandarizar”!

4. Pipelines y Modelos de Datos (La Construcción)

Construir el *pipeline* es la parte más compleja del proyecto.

- **Evitar “Hervir el Océano”:** No intentes modelar todos los datos de una vez. Empieza con algo pequeño y mantenlo flexible. El error más común es saltar a la tecnología sin pensar en los requisitos.
- **Recordar la Regla del 80/20:** El 80% del tiempo de un científico de datos se dedica a encontrar, limpiar y organizar los datos, y solo el 20% al desarrollo del modelo.
- **Construir Pipelines Modulares e Inmutables:** La mejor práctica es diseñar los *pipelines* como una serie de pasos de transformación **aislados e inmutables**. Cada paso toma un conjunto de datos como entrada y produce uno nuevo como salida, sin modificar el original. Esto facilita la reutilización, las pruebas y la depuración.
- **SQL vs. NoSQL Pipelines:**
 - **SQL:** Mejor para consultas complejas y **joins**, pero requiere una estructura de datos rígida y más pasos de validación.
 - **NoSQL:** Más simple y flexible, más fácil de escalar horizontalmente, pero más difícil para consultar e integrar datos.
- **El Linaje es Obligatorio:** Todo el *pipeline* debe generar metadatos de **linaje**. Debes poder rastrear cada dato en tu DDS hasta el producto de datos original del que provino. Esto es crucial para la calidad, la depuración y la gobernanza.

Estas mejores prácticas buscan guiar al lado consumidor para que pueda crear valor de forma ágil y flexible, pero sin caer en el caos, aprovechando los principios de estandarización, reutilización y gobernanza federada.

What the fuck is a pipeline?

En el contexto del libro de Strengtholt, y especialmente cuando hablamos de los DDS, un **pipeline de datos** es una **serie de pasos automatizados que extraen datos**

de uno o más productos de datos, los transforman, y los cargan en un nuevo destino (el Domain Data Store) como un conjunto de datos integrado y listo para usar.

La Analogía: La Línea de Ensamblaje de una Fábrica

La mejor manera de pensarlo es como una **línea de ensamblaje en una fábrica**:

1. **Materia Prima (Entrada)**: Al principio de la línea llegan las materias primas. En nuestro caso, son los **productos de datos** (ej: la tabla de “Compras”, la tabla de “Tickets de Soporte”).
2. **Estaciones de Trabajo (Pasos)**: La materia prima pasa por diferentes estaciones, donde las máquinas realizan operaciones específicas. Estos son los **pasos del pipeline** (las transformaciones).
3. **Producto Terminado (Salida)**: Al final de la línea sale un producto terminado y listo para usar. En nuestro caso, es la **tabla final dentro del DDS** (ej: la tabla `training_dataset` para el modelo de *churn*).

El pipeline es **todo el proceso automatizado**, desde que se recogen los ingredientes hasta que sale el producto final.

¿Qué Hace Exactamente un Pipeline? ¿Cuál es su Función?

La función principal de un pipeline en el lado del consumidor es **operacionalizar la lógica de negocio compleja**. Es el “cómo” se convierten los datos crudos en *insights* de forma repetible y confiable.

Un pipeline realiza las siguientes acciones clave:

- **Mueve Datos (Extraer y Cargar - EL)**: Se conecta a los productos de datos de origen, extrae la información necesaria y la copia en el entorno del consumidor (el DDS).
- **Limpia y Estandariza Datos**: Aplica reglas para corregir errores, manejar valores nulos, estandarizar formatos, etc.
- **Combina Datos (JOIN)**: Une información de múltiples productos de datos para crear una vista unificada.

- **Transforma Datos (Transform - T):** Aplica todas las transformaciones que discutimos antes:
 - **Agregaciones (GROUP BY):** Cambia la granularidad de los datos (de transacciones a clientes).
 - **Feature Engineering:** Crea nuevas columnas y características que son vitales para el análisis o el modelo de ML.

Flujo de Trabajo de un Pipeline en la Vida Real (Ejemplo del Modelo de Churn)

Imaginemos que el equipo de Ciencia de Datos ha construido el pipeline para crear la tabla `training_dataset` para su modelo de *churn*.

Objetivo del Pipeline: Crear y actualizar diariamente la tabla `training_dataset` en el DDS del equipo.

Así es como funcionaría:

1. **Disparo (Trigger):** Un orquestador de tareas (como Apache Airflow) activa el pipeline automáticamente todos los días a las 3:00 AM.
2. **Paso 1: Ingesta (Copia de Datos)**
 - El pipeline se conecta a los productos de datos: `pedidos.compras`, `soporte.tickets`, y `producto.actividad_app`.
 - Ejecuta `SELECT * FROM ... WHERE fecha > ayer` para extraer solo los datos nuevos de cada fuente.
 - Carga estos datos en tablas “temporales” o “de staging” dentro de su DDS.
3. **Paso 2: Transformación (Unión y Limpieza)**
 - El pipeline ejecuta un `JOIN` complejo para unir las tres tablas temporales por `customer_id`.
 - Aplica pasos de limpieza (ej: convierte todas las fechas a un formato estándar).
 - El resultado se guarda en otra tabla intermedia.

4. Paso 3: Transformación (Agregación y Lógica de Negocio)

- El pipeline toma la tabla intermedia, la agrupa por `customer_id` y calcula las *features* clave: `total_gastado`, `numero_de_tickets`, `dias_desde_ultima_sesion`, etc.
- Esta es la lógica de negocio más valiosa.

5. Paso 4: Carga Final

- El pipeline toma el resultado del paso anterior y lo inserta o actualiza en la tabla final: `datascience_churn_project.training_dataset`.
6. **Finalización y Alertas:** El orquestador marca el pipeline como “exitoso” y envía una notificación. Si algo falla en cualquier paso, marca el pipeline como “fallido” y alerta al equipo.
-

Dos Tipos de Pipelines

Es útil distinguir que en esta arquitectura hay dos tipos de pipelines con funciones diferentes:

- **Pipeline del Proveedor:** Es el que construye el **producto de datos**. Su objetivo es tomar datos brutos de un sistema operacional y producir un conjunto de datos limpio y confiable.
- **Pipeline del Consumidor (el de este capítulo):** Es el que construye la **solución de negocio**. Su objetivo es tomar uno o más productos de datos y producir un *insight* o un conjunto de datos integrado en un DDS.

En resumen, el **pipeline** es el código automatizado que ejecuta todas las transformaciones necesarias para convertir los productos de datos en bruto en los datos integrados y de alto valor que viven dentro del DDS. Es el corazón operativo del lado del consumidor.