Data Reliability Engineering

Summary

Concepts

The first part of this book exposes the different concepts around the data reliability engineering subject. It's intended to be heavilly technical, in contrast with the subsequent parts, intended to explore practical use cases.

Introduction to Systems and Systems Reliability

These chapters explore what are systems, what is reliability, and how to understand systems reliability, specially its impediments, its attributes, and mechanisms to design and maintain reliable systems. All this for general systems, data systems, and data products.

Data Quality

This chapter explores what is data, what is quality, and what is data quality, to finally explore what is data reliability. The goal is to understand these concepts in all aspects of the data: life cycle, design, modelling, governance, management, access, security, uses, legal frameworks, best practices, maturity, standards, etc.

Processes

This chapter explores, for a given system, the concept of data processes, data and information flow, workflows, orchestration, pipelines, ETL, and ELT.

Operations

This chapter explores the concept of SRE, DataOps, DevOps, Agile methodologies, CI/CD, and other methodologies to assure reliable data operations.

Data Architecture

This chapter explores what is data architecture, including its sources, its storage (Data Lake, Data Warehouses, Data Marts), its application (OLAP servers, processing engines), and its presentation (dashboards, reports).

Confiabilidad de Sistemas

> La confiabilidad de un sistema es la propiedad del sistema que permite calificar, justificadamente, como fiable al servicio que proporciona.

El objetivo de este capítulo es introducir los conceptos de confiabilidad y seguridad explorados por Alan Burns y Andy Wellings en su libro "Sistemas de Tiempo Real y Lenguajes de Programación", conceptos desarrollados por diferentes industrias principalmente entre los años 60 y 90, y los conceptos de Site Reliability Engineering (SRE) desarrollados a partir de los 2000, además de complementarlo con conceptos de confiabilidad trabajados en otras ingenierías (mecánica, industrial, etc.), bien cómo contextualizarlo con conceptos trabajados actualmente en la industria de software y sistemas informáticos.

Impedimentos

> Los impedimentos impiden el perfecto funcionamento de un sistema, o son consecuencia de este. En ese subcapítulo se trabajará la detección de los diferentes tipos de impedimentos, los cuales, los **Fallos**, **Errores** y **Defectos**.

Atributos

> El modo y las medidas mediante las cuales se puede **estimar la calidad de un servicio confiable**.

Mecanismos

Los mecanismos los cuales se trabaja la confiabilidad de sistemas, sea por interiorización y adopción de buenas prácticas, sea por la aplicación de metodologías, arquitecturas o herramientas específicas. Este subcapítulo busca formar un **marco de trabajo** el cual los ingeníeros puedan adoptar la confiabilidad de sistemas desde el mismo diseño.

Impedimentos

Fallos, Errores y Defectos

> Los **fallos** son el resultado de problemas internos no esperados que el sistema manifiesta eventualmente en su comportamiento externo. Estos problemas se llaman **errores**, y sus causas mecánicas o algorítmicas se denominan **defectos**. Cuando el comportamiento de un sistema se desvía del especificado para él, se dice que es un fallo.

Los sistemas están compuestos de **componentes**, cada uno de los cuales se puede considerar como un sistema en sí mismo. Así, un fallo en un sistema puede inducir un defecto en otro, el cual puede acabar en un error y en un fallo potencial de este sistema. Esto puede continuar y producir un efecto en cualquier sistema relacionado, y así sucesivamente.

Un componente defectuoso de un sistema es un componente que producirá un error bajo un conjunto concreto de circunstancias durante la vida del sistema. Visto en términos de transición de estados, *un sistema puede ser considerado como un número de estados externos e internos*.

Un estado externo no especificado en el comportamiento del sistema se considerará un fallo del sistema. El sistema en sí mismo consta de un número de componentes (cada uno con sus propios estados), contribuyendo todos ellos al comportamiento externo del sistema. La combinación de los estados de estos componentes se denomina estado interno del sistema. *Un estado interno no especificado se considera un error, y el componente que produjo la transición de estados ilegal se dice que es defectuoso*.

Definiré tres tipos de fallos:

- **Fallos transitorios**: comienza en un instante de tiempo concreto, se mantiene en el sistema durante algún periodo de tiempo, y luego

desaparece.
- **Fallos permanentes**: comienzan en un instante determinado y permanecen en el sistema hasta que son reparados.
- **Fallos intermitentes**: son fallos transitorios que ocurren de vez en cuando.

Modos de Fallos

> Un sistema puede fallar de muchas maneras. Un diseñador puede diseñar el sistema suponiendo un número finito de modos de fallo, sin embargo el sistema puede fallar de manera diferente a lo esperado.

Podemos clasificar los modos de fallos de los servicios que proporciona un sistema, los cuales:

- **Fallos de valor**: el valor asociado con el servicio es erróneo.
- **Fallo de tiempo**: el servicio se completa a destiempo.
- **Fallo arbitrario**: combinación de fallos de valor y tiempo.

Los modos de fallo de valor se denominan **domínio de valor**, y son clasificados en **error de límites**, y **valor erróneo**, dónde el valor se encuentra fuera del rango estipulado.

Los fallos en el dominio del tiempo pueden hacer que el servicio sea entregado:

- **Demasiado pronto** (adelantado): el servicio se entrega antes de lo requerido.
- **Demasiado tarde** (retrasado o error de prestaciones): el servicio se entrega después de lo requerido.
- **Infinitamente tarde** (fallo de omisión): el servicio nunca es entregado.
- **No esperado** (fallo de encargo o improvisación): el servicio es entregado sin ser esperado.

En general, podemos suponer los modos que un sistema puede fallar:

- **Fallo descontrolado**: un sistema que produce errores arbitrales, tanto en el dominio del valor como en el del tiempo (incluyendo errores de improvisación).
- **Fallo de retraso**: un sistema que produce servicios correctos en el dominio del valor, pero que sufre errores de retraso en el tiempo.
- **Fallo de silencio**: un sistema que produce servicios correctos tanto en el dominio del valor como en el del tiempo, hasta que falla. El único fallo posible es el de omisión, y cuando ocurre, todos los servicios siguientes también sufrirán fallos de omisión.
- **Fallo de parada**: un sistema que tiene todas las propiedades de un fallo silencioso, pero que permite que otros sistemas puedan detectar que ha entrado en el estado de fallo de silencio.
- **Fallo controlado**: un sistema que falla de una forma especificada y controlada.
- **Sin fallos**: un sistema que siempre produce los servicios correctos.

Atributos

Confiabilidad

> Es la probabilidad *R(t)* de que el sistema **siga funcionando al final del proceso**. El tiempo *t* se mide en horas continuas de trabajo entre diagnósticos. La tasa constante de fallos λ se mide en *fallos/h*. La vida útil de un componente del sistema es la región constante (escala logarítmica) de la curva entre vida del componente (Component Age) y su tasa de fallos (Failure Rate). La región de la gráfica antes del equilibrio es el Burn In Phase, y la región en donde la tasa de fallos empieza a crecer es el End of Life Phase. Así tenemos $R(t)=exp(-\lambda t)$.

Disponibilidad

> Es la medida de la **frecuencia de los periodos de servicio incorrecto**.

Fiabilidad

> Continuidad de entrega del servicio.

Es una medida (probabilidad) del **éxito con el que el sistema se ajusta a la especificación definitiva de su comportamiento**.

Seguridad

> Es la ausencia de condiciones que pueden causar daños y propagación de **daños catastróficos** en producción.

Sin embargo, como esa definición puede clasificar cómo inseguros virtualmente cualquier proceso, consideraremos a menudo el término **percance**.

> Un percance es un **evento no planeado** o secuencia de eventos que pueden producir daños catastróficos.

Por mayores que sean su similitud con la definición de **fiabilidad**, se debe considerar la diferencia en su énfasis. La fiabilidad es la medida de éxito con la cual el sistema se ajusta a la especificación de su comportamiento, normalmente en términos de **probabilidad**. La seguridad, sin embargo, es la **improbabilidad de que se den las condiciones que conducen a un percance, independientemente si se realiza la función prevista**.

Integridad

> Es la ausencia de condiciones que pueden llevar a alteraciones inapropiadas de los datos en producción. Es la **improbabilidad de que se den las condiciones que alteran en producción datos inapropiados, independientemente si se realiza la función prevista**.

Mantenimiento

Capacidad de superar reparaciones y evolucionar.

Escalabilidad

Capacidad de adecuación al negocio.

Deficiencias

Circunstancias que causan o son producto de la no **confiabilidad**.

Mecanismos

Prevención de Fallos : Evitación

Tolerancia de Fallos

Prevención de Fallos: Eliminación

Predicción de Fallos

Herramientas de Confiabilidad

Prevención de Fallos : Evitación

Existen dos fases en la prevención de fallos: **evitación y eliminación**.

> Con la evitación se intenta limitar la introducción de datos y objetos potencialmente defectuosos durante la ejecución del proceso.

Como:

- La utilización de fuentes de información validadas y limpias (cuando sea posible).
- La introducción de procesos de limpieza y validación de los datos (data cruda).
- La introducción de validación de disponibilidad de tablas y columnas, y la introducción de operadores de rama para su manejo.

Tolerancia de Fallos

> Debido a las limitaciones en las prevenciones de fallos, una vez que la data y los procesos cambian con frecuencia, es necesario recurrir a la tolerancia de fallos.

Existen diferentes niveles de tolerancia a fallos:

- **Tolerancia total**: no hay manejo de condiciones adversas o no deseadas, el proceso no se adapta a las validaciones y variables de entorno u otras informaciones externas para la ejecución de los tasks.
- **Degradación controlada** (o caída suave): notificaciones serán disparadas en la presencia de fallos, y siendo el suficiente importantes para interrumpir el flujo de tareas (thresholds, inexistencia o indisponibilidad de la data), los operadores de rama seleccionarán los tasks subsiguientes.
- **Fallo seguro**: los fallos detectados son suficientes para detectar que el proceso no debe ocurrir, un operador de cortocircuito cancela la ejecución de los subsiguientes tasks, los responsables son notificados, y caso no exista un proceso automático para lidiar con el problema, el equipo de data puede tomar acciones como volver a ejecutar los procesos que generan los inputs necesarios, o escalar el caso.

El diseño de los procesos tolerantes a fallos supone:

- Los algoritmos de las tareas se han diseñado correctamente.
- Se conocen todos los posibles modos de fallos de los componentes.
- Se han tenido en cuenta todas las posibles interacciones entre el proceso y su entorno.

Redundancia

> Todas las técnicas utilizadas para conseguir tolerancia a fallos se basan en añadir elementos externos al sistema para que detecte y se recupere de fallos. Estos elementos son redundantes en el sentido de que no son necesarios para el normal funcionamiento del sistema, a esto llamamos **redundancia protectora**. El objetivo de la tolerancia es minimizar la redundancia, maximizando la fiabilidad, siempre bajo las restricciones de complejidad y tamaño del sistema. **Se debe tener cuidado al diseñar los sistemas tolerantes a fallos, ya que los componentes incrementan la complejidad y mantenimiento de todo el sistema, lo que puede en sí, conducir a sistemas menos fiables**.

Clasificamos la redundancia en los sistemas en estáticas y dinámicas. La **redundancia estática**, o enmascaramiento, consiste en que los componentes redundantes son utilizados para ocultar los efectos de los fallos. La **redundancia dinámica** es la redundancia aportada dentro de un componente que hace que el mismo indique, implícita o explícitamente, que la salida es errónea; la recuperación debe ser proporcionada por otro componente. Esta técnica de tolerancia a fallos tiene cuatro fases:

1. **Detección de errores**: no se utilizará ningún esquema de tolerancia a fallos hasta que se haya detectado un error.
2. **Confinamiento y valoración de daños**: cuando se detecte un error, debe estimarse la extensión del sistema que ha sido corrompida (diagnóstico de error) y su escopo.
3. **Recuperación del error**: este es uno de los aspectos más importantes de la tolerancia a fallos. Las técnicas de recuperación de errores deberían dirigir al sistema corrupto a un estado a partir del cual pueda continuar con su normal funcionamiento (quizás con una degradación funcional).
4. **Tratamiento del fallo y continuación del servicio**: un error es un síntoma de un fallo; aunque el daño pudiera haber sido reparado, el fallo continúa existiendo, y por lo tanto el error puede volver a darse a menos que se realice algún tipo de mantenimiento.

1. Detección de errores

> La efectividad de un sistema tolerante a fallos depende de la **efectividad de detección de errores**.

La detección de errores se clasifican en:

- **Detecciones en el entorno**. Los errores se detectan en el entorno en el cual se ejecutan el programa. Son manejados por las excepciones (exceptions).
- **Detección en la aplicación**. Los errores se detectan en la misma aplicación.
    - **Comprobaciones inversas**. Aplicadas en componentes de relación isomórfica (uno a uno) entre la entrada y la salida. En este método, se toma la salida y se calcula la entrada, lo cual es comparado con el valor de entrada original. Para casos de números reales, es necesario adoptar técnicas de comparación inexactas.
    - **Comprobación de racionalidad**. Se basan en el conocimiento del diseño y de la construcción del sistema. Comprueban que el estado de los datos o el valor de un objeto es razonable basándose en su supuesto uso.

2. Confinamiento y valoración de los daños

> Siempre existirá una magnitud de tiempo, entre la ocurrencia de un defecto y la detección del error, siendo por lo tanto importante la valoración de cualquier daño que se haya podido producir en este intervalo de tiempo.

Aunque el tipo de error detectado podrá dar ideas sobre el daño a la rutina de tratamiento del error, podrían haber sido diseminadas informaciones erróneas por el sistema y su entorno. Así, la valoración de los daños estará directamente relacionada con las precauciones tomadas por el diseñador de este sistema para el confinamiento del daño (**construcción de cortafuegos**).

El confinamiento del daño se refiere a la estructuración del sistema de modo que se minimicen los daños causados por un componente defectuoso.

Existen dos técnicas principales para estructurar los sistemas de modo que se facilite el confinamiento de daños: **descomposición modular** y **acciones atómicas**. Por descomposición modular entiéndase que los sistemas deben ser descompuestos en componentes, cada uno de los cuales se representa por uno o más módulos. La interacción de los componentes se produce a través de interfaces bien definidas, y los detalles internos de los módulos están ocultos y no son accesibles directamente desde el exterior. Esto hace más difícil que un error en un componente pase indiscriminadamente a otro.

La descomposición modular proporciona al sistema una estructura estática, ya las acciones atómicas proporcionan al mismo una estructura dinámica. Se dice que una acción es atómica si no existen interacciones entre la actividad y el sistema durante el transcurso de la acción. Estas acciones se utilizan para mover el sistema de un estado consistente a otro, y para restringir el flujo de información entre los componentes.

3. Recuperación de errores

> Una vez detectada la situación de error, y que sus posibles daños hayan sido valorados, se inician los procedimientos de recuperación de errores. Esta fase es probablemente la más importante dentro de las técnicas de tolerancia a fallos, la cual debe transformar un estado erróneo del sistema en otro desde el cual el sistema pueda continuar con su funcionamiento normal, quizás con una cierta degradación en el servicio.

Aquí citaré dos estrategias para la recuperación de errores: recuperación **hacia adelante**, y **hacia atrás**. La recuperación de errores hacia adelante intenta continuar desde el estado erróneo realizando correcciones selectivas en el estado del sistema, que incluye proteger cualquier aspecto del entorno controlado que pudiera ser puesto en riesgo o dañado por el fallo.

La recuperación hacia atrás se basa en restaurar el sistema a un estado seguro previo a aquél en el que se produjo el error, para luego ejecutar una sección alternativa de la tarea. Ésta tendrá la misma funcionalidad que la sección que produjo el defecto, pero utilizando un algoritmo distinto. Se espera que esta alternativa no produzca el mismo defecto que la versión anterior, así que dependerá del conocimiento del diseñador sobre los posibles modos de fallo de este componente.

El diseñador debe tener claro los niveles de degradación de un servicio, teniendo en cuenta los servicios y procesos que dependen de éste. La recuperación de errores hace parte de los procesos de acción correctiva y acciones preventivas (CAPA - Corrective Action and Preventive Action Process), el cual se trabajará en dos momentos: en ese mismo capitulo de tolerancia a fallos cuando se trabajen las acciones correctivas, y en el próximo capitulo, prevención de fallos, cuando se aborde el tema de acciones preventivas.

4. Tratamiento de los fallos y servicio continuado

> Un error es una manifestación de un defecto, y aunque la fase de recuperación del error puede haber llevado el sistema a un estado libre de error, el error se puede volver a producir. Por lo tanto, la fase final de la tolerancia de fallos es erradicar el fallo del sistema, de forma que se pueda continuar con el servicio normal.

Prevención de Fallos: Eliminación

La segunda fase de prevención de fallos es la eliminación de fallos. Consiste normalmente en procedimientos para encontrar y eliminar las causas de los errores. Aunque se pueden utilizar técnicas como los revisores de código (IDEs, linter) y el debugging en local, ni siempre se llevan a cabo las revisiones por pares y pruebas exhaustivas con las distintas combinaciones de estados de entrada y entorno.

Las pruebas en QA no pueden verificar que los valores de salida sean compatibles con el negocio y sus aplicaciones, así que se concentran normalmente en modos de fallo de tiempo (timeouts) y **defectos**. Desafortunadamente, la prueba del sistema no puede ser exhaustiva y eliminar todos los potenciales fallos, principalmente por:

- Las pruebas se utilizan para demostrar la presencia de fallos, no su ausencia.
- La dificultad de realizar pruebas en producción. La manera de probar fallas en producción son del tipo **combate real**, o sea, la consecuencia de los errores pueden afectar directamente el negocio, haciendo que pueda tomar malas decisiones (ejemplo: un mal cálculo de un KPI, puede además de llevar a acciones erróneas, disminuir la confianza del negocio en los procesos de data). Existen alternativas de diseño de procesos para detección de fallos en producción, las cuales discutiré más adelante.
- Los errores que han sido introducidos en la etapa de requisitos del sistema puede que no se manifiesten hasta que el sistema esté operativo. Ejemplo: Un DAG para el procesamiento y limpieza de la data de delivery de las campañas de marketing online que se diseñó para ejecutar antes del proceso de atribución de media, a las 4am, no llevó en consideración que la data no estará disponible hasta las 5am.

Predicción de Fallos

La predicción acurada y rápida de los fallos posibilita a los que mantenemos los procesos asegurar mayor disponibilidad de los servicios. Desafortunadamente, la predicción de fallos es muchísimo menos sencilla que su detección.

Para poder predecir un fallo, éste debe ser identificado y clasificado. Los fallos también deben ser predecibles (o capaces de predicción), lo que significa que existen alteraciones de estados de los sistemas (y componentes) que llevan al fallo, o el fallo ocurre regularmente siguiendo algún patrón. Ambos casos pueden ser traducidos a problemas de predicción de series temporales, y la data de los sensores y logs puede ser trabajada para entrenar los modelos de predicción.

La data colectada muy difícilmente estará lista para ser utilizada por los modelos de predicción, así que uno o más tareas de preprocesamiento deben llevarse a cabo:

- **Sincronización de la data**: las métricas colectadas por diversos agentes (sensores) deben alinearse en la dimensión de tiempo.
- **Limpieza de la data**: remoción de data innecesaria, y generación de data faltante (ej: interpolación).
- **Normalización de la data**: los valores de las métricas son normalizados para que las magnitudes sean comparables.
- **Selección de features**: las métricas relevantes son identificadas para su utilización en los modelos.

Una vez preprocesada la data, la misma será utilizada en dos pipelines: pipeline de entrenamiento, y pipeline de inferencia. El pipeline de entrenamiento usa la data en bulk, para entrenar el modelo a ser disponibilizado al pipeline de inferencia. Los resultados de la inferencia indicará la existencia o no de tipos específicos de fallas, sobre la muestra de métricas monitoreadas.

Herramientas de Confiabilidad

Diagramas de blocos de confiabilidad

> Los diagramas de blocos de confiabilidad (reliability block diagram, o RBD) es un método para diagramar e identificar como la confiabilidad de componentes (o subsistemas) *R(t)*, contribuyen para el éxito o fracaso de una redundancia. Es decir, es un método que puede ser utilizado para diseñar y optimizar componentes y seleccionar redundancias, visando bajar los failure rates.

Un RBD es representado en una serie de blocos conectados (en série, en paralelo, o su combinación), indicando componentes redundantes, indicando el tipo de redundancia y su respectivo failure rate.

Al analizarse el diagrama, se indican componentes que fallaron y los que no fallaron. Si es posible encontrar una ruta o camino entre el inicio y el fin de proceso con componentes que no fallaron, se puede suponer que el proceso se puede completar.

Cada RBD debe incluir afirmaciones o sentencias listando todas las relaciones entre los componentes, es decir que condiciones llevarón a tomar un componente u otro en la ejecución del proceso.

Links:

- Université Angers
- Wikipedia
- HPReliability
- Sydney Water

Failure Reporting, Analysis, and Corrective Action System (FRACAS)

> FRACAS es un sistema o proceso definido para el reporte, clasificación y análisis de fallos, bien como la planeación de acciones correctivas de dichos fallos. Es parte del proceso guardar el historial de los análisis y acciones tomadas.

Llevar a cabo dicho proceso supone automatizar el análisis de los logs de los procesos de data (logs), commits, pull requests y tickets.

La implementación de proceso es cíclica y se da por (FRACAS Kaizen Loop adaptado):

- **Failure Mode Analysis**: analysis de los modos de fallos.
- **Failure Codes Creation**: creación de códigos de fallos, o la metodología para clasificarlos.
- **Work Order History Analysis**: análisis del historial de tickets enviados al equipo de data.
- **Root Cause Analysis**: análisis de las causas raíces.
- **Strategy Adjustment**: ajuste de estratégia.

Links:

- Wikipedia
- Reliability Web
- IEEE Best Practices

Spare Parts Stocking Strategy

> Con suerte siempre existirán disponibles fuentes de datos limpias, con complejas transformaciones y limpiezas, que ahorran tiempo y procesamiento, y que pueden ser usadas en multiples etapas de

> multiples procesos, sin embargo las mismas pueden temporalmente estar no disponibles o fallar. Una vez identificados tales fuentes, y constatado que son críticas a un sistema o proceso, es prudente tener tareas mínimas de limpieza y transformaciones que trabajen sobre los datos crudos o fuentes de la fuente, que quizas no resultará en datos finales con los mismos niveles de detalles, pero que serán lo suficiente buenos.

Tales tareas no son diseñadas para hacer parte del flujo normal de los procesos, pero son "piezas de recambio", disponibles para cuando los tiempos de mantenimiento son demasiado largos. El empleo de dichas tareas deben ser por el mínimo de tiempo posible, mientras el equipo tiene tiempo de resolver los fallos en la tarea original, o diseñar su reemplazo.

Links:

- ReliabilityWeb

Availability Controls

> Fallos de disponibilidad pueden ocurrir por un sin numero de razones (desde hardware hasta bugs), y algunos sistemas o procesos tienen suficiente relevancia para que controles de disponibilidad (availability controls) sean implementados, para asegurar que determinados servicios o data sigan disponibles cuando ocurra dichos fallos.

Los controles de disponibilidad van desde el uso de backups periódicos de la data, snaps, timetravel, procesos redundantes, sistemas de respaldo en servidores locales o cloud, etc.

Links:

- WhiteHatSec
- Law Insider

- Control Global

Acciones Correctivas

> Parte del CAPA (Corrective Action and Preventive Action Process), las acciones correctivas (CAP - Corrective Action Process) consisten en la detección de fallos, la determinación de sus causas raíces, las acciones de corrección, y la toma de medidas de prevención para que el mismo fallo vuelva a ocurrir por los mismos motivos. La definición completa se encuentra en la ISO 9001.

Diversas herramientas y técnicas son utilizadas en diversas industrias para su aplicación, dentre ellas, PDCA (Plan, Do, Check, Act), DMAIC (Define. Measure, Analyse, Improve, Control), 8D, etc. De manera general cualquier herramienta, técnica o metodologia, es sumarizada en la ISO 9001, en siete "pasos":

1. Definir el problema. Consiste en definir que el problema sea real, identificar Quien, Qué, Cuando, Dónde y Por qué. En el mundo de la ingeniería de datos, ese paso debe ser, en lo posíble, automático, y el fallo debe ser detectado desde sensores.
2. Definir el escopo. Consiste en mensurar el problema a se resolver, conociendo su frecuencia, a que procesos o tareas, y stakeholders afecta. Para los procesos de data, muchos de los detalles de escopo ya deberían ser información conocida, desde el diseño de los procesos y tareas, ya la frecuencia puede ser levantada desde los procesos de observability y FRACAS.
3. Acciones de confinamiento o contención. Son medidas puntuales y adoptadas por el mínimo de tiempo posible, mientras se trabaja en la solución definitiva del fallo. De antemano, tales medidas ya deberían estar diseñadas, para cada tarea o sub-tarea. La selección de medidas debería estar automatizada, de no serlo, se deben implementar de inmediato.
4. Identificación de causas raíz. Diagnosis clara, precisa y completa del fallo. Su documentación hace parte del FRACAS.

5. Planeación de acciones correctivas. Consiste en la planeación de acciones de corrección basadas especificamente en la causa raíz.
6. Implementación de acciones correctivas. Consiste en la implementación final de las acciones correctivas en el proceso, que deben automaticamente estar disponibles cuando fallos similares se presenten.
7. Acompañamento de los resultados (Follow up). Documentación, comunicación, FRACAS completos.

Antifragilidad

> Inspirado en el libro *Antifragile: Things That Gain from Disorder* de Nissim Nicholas Taleb, la antifragilidad difere de los conceptos de resiliencia o robustez, dónde los sistemas buscan mantener su nivel de confiabilidad, sino que desde su diseño, los sistemas aumentar su confiabilidad con respecto a los inputs del sistema.

La antifragilidad propone un cambio de diseño de los sistemas (en el escopo de este libro, procesos), los cuales comumente son diseñados para ser frágiles, en el sentido de que si el mismo opera fuera de sus requerimientos, lo mismo fallará. La antifragilidad propone lo contrário, diseñar sistemas que se vuelven mejores cuando expuestos a cargas fuera de los requerimientos. En ese sentido, los sistemas no son diseñados para responder solamente a lo esperado o anticipado, sino que interactúan con su entorno en tiempo real y se adaptan a ello.

- Self-healing
- Real time sensoring, monitoring
- Live FRACAS
- System Health Management
- Automatic Repair

Links:

- NASA

- Refuses

Bulkhead Pattern

> En el mundo náutico encontramos los mamparos, placas de madera que encontramos en los barcos, que buscan que el barco no naufrague cuando se tiene comprometida una porción del casco. Para los sistemas el Bulkhead Pattern adapta exactamente esa idea, de que un fallo en una porción del sistema no comprometa el sistema en su totalidad.

Este design pattern es aplicado comumente en el desarrollo de software, consiste en no sobrecargar un servicio con más llamadas de las que puede soportar en un determinado tiempo, un ejemplo de eso es Hystrix, de Netflix.

Links:

- Netflix

Calidad de Datos

Fundamentos de la Calidad de Datos

Ciclo de vida de los datos

DAMA

POSMAD

COBIT

Gobierno versus Gestión de los datos

Datos Maestros

Arquitectura MDM

Modelo de madurez

Estándares

ISO 8000

ISO/IEC 22745

Calidad de los procesos de datos

DAMA DMBOK

Modelo de Aiken

Data Management Maturity Model (DMM)

Modelo IBM

Modelo de Gartner

TQDM

DCAM

Modelo MAMD

Modelos de calidad de datos

Modelo de calidad de datos

Medidas de calidad de datos

Proceso de evaluación

Confiabilidad de Datos

> Tomándose por base los fundamentos y metodologías desarrolladas por los **Site Reliability Engineers**, conocidos como los “firefighters” del mundo de la ingeniería de sistemas, los cuales construyen sistemas automatizados para optimizar la disponibilidad de las aplicaciones (reducir downtime), definiremos **Data Reliability** cómo la capacidad del equipo de data en entregar alta disponibilidad de la data durante todo el ciclo de vida de la misma. En resumen, garantizar los periodos de tiempo que la data no presenta inacurácia, no es faltante ni errónea.

...

Consecuencias de la confiabilidad

Consecuencias (en administrar el downtime de la data):

- Los equipos de data reducen de manera muy importante el tiempo perdido en “apagar incendios”, escalaciones y troubleshooting de la data. Utilizan ese tiempo para enfocarse en la construcción de una buena infraestructura, y en agregar valor a la data.
- Los equipos de data son más rápidos en actualizar y modificar la infraestructura de la data, ya que tienen claro la confiabilidad del sistema.
- Los equipos de data ganan el respeto y la confianza de los stakeholders, ya que entregan datos confiables de manera consistente.

Data Architecture

> Data Architecture is about how the data is managed, from **collection**, **transformations**, **distribution**, and **consumption**.

It includes the models, policies, rules, and standards that govern which data is collected and how it is stored, arranged, integrated, and put to use in data systems and in organizations.

The data architecture aims to set standards to all data systems, and the interaction between them. It also dictates and materialises the organization understanding of its business in a Conceptual, Logical, and Physical level.

The Zachman Framework for enterprise architecture, understands the data architecture in five layers:

1. Scope/Contextual: subjects and architectural standards important for the business.
2. Business/Conceptual: business entities, its attributes and associations.
3. System/Logical: how entities are related.
4. Technology/Physical: representation of a data design as implemented in a database management system.
5. Detailed Representations: databases.

Conceptual Layer

> Represents all **Business Entities**.

The **Conceptual Data Model** (CDM) consists of the **Business Entities**, like **User**, **Branch**, **Product**. The business entities (or business objects) carry *attributes* (name, identifiers, timestamps, etc.), and *associations* (relationships)

with other business entities. The complete set of business entities represents the business relationships.

From a data architecture perspective, these business entities are represented in a **Conceptual Schema**, which consists of a map of **concepts** (business entities) and their **relationships** in a database, normally in a **Data Structure Diagram** (DSD). It may also include **Enterprise Data Modelling** (EDM) outputs like entity–relationship diagrams (ERDs), XML schemas (XSD), and an enterprise wide data dictionary.

The conceptual schema describes the semantics of an organization, and represents a series of assertions about its nature. It describes the objects of significance (business entities), of which the organization is ineterested in collecting information of, its characteristics (attributes), and the associations between each pair of objects of significance (relationships). Please note that it's not the actual database design, and it is represented in different abrastraction layers.

Examples:

- Each ORDER must be from one and only one USER.
- Each ORDER contains one or more PRODUCTS.
- Each ORDER contains products from one or more BRANCHES.

Logical Layer

> Represents the logic and how the entities are related.

The **Logical Data Model** (LDM), also known as Domain Model, represents the abstract structure of a domain of information, expressed independently of a particular database management product or storage technology (physical data model), but in terms of data structures such as relational tables and columns, object-oriented classes, or XML tags.

Once validated and approved, the logical data model can become the basis of a physical data model and form the design of a database.

Physical Layer

> The representation of a data design as implemented, or intended to be implemented, in a database management system.

The **Physical Data Model** (PDM) typically derives from a logical data model (LDM), though it may be reverse-engineered from a given database implementation. A complete physical data model will include all the database artifacts required to create relationships between tables or to achieve performance goals, such as indexes, constraint definitions, linking tables, partitioned tables or clusters.

CDM vs LDM vs PDM

Data Constructs

- CDM: uses general high-level data constructs from which Architectural Descriptions are created in non-technical terms.
- LDM: includes entities (tables), attributes (columns/fields) and relationships (keys). Is independent of technology (platform, DBMS).
- PDM: includes tables, columns, keys, data types, validation rules, database triggers, stored procedures, domains, and access constraints, as primary keys and indices for fast data access.

Naming Conventions

- CDM: non-technical names, so that executives and managers at all levels can understand the data basis of Architectural Description.
- LDM: uses business names for entities & attributes.

- PDM: uses more defined and less generic specific names for tables and columns, such as abbreviated column names, limited by the database management system (DBMS) and any company defined standards.

Modern Data Architecture

A modern approach to data architecture, extensivelly adapted by StartUps, reduces, restricts or understands the data architecture as the implementation of a Data Warehouse, a Data Lake + Data Warehouse, or a Data Lakehouse architecture, consisting of the data sources plus two or three tiers:

- Bottom/Data Tier: data warehouse server with functional gateway (ODBC, JDBC, etc.). May also include the data lake.
- Middle/Application Tier: houses the business logic used to process user inputs (OLAP Servers, Snowflake, Apache Redshift, Databricks Data Lakehouse Platform, Apache Spark, etc.).
- Top/Presentation Tier: front-end tools (Power BI, Tableau, etc.).

Data Lake

> A Data Lake is a data repository for storing large amounts of Structured, Semi-Structured, and Unstructured data.

It is a repository for storing all types of data in its native format without fixed limits on account size or file. Data Lake stores a high data quantity to increase native integration and analytic performance. The Data Lake democratizes data and provides a cost-effective way of storing all organization data for later processing.

Data Lake vs Data Warehouse

> A Data Warehouse is a repository that exclusively keeps pre-processed data from a Data Lake or many databases.

ETL operations are used to arrange data in multi-dimensional structures so that Analytics workflows using Data Warehouses can be accelerated. Business Intelligence specialists and Data Analysts can generate reports and develop dashboards using the data housed in a Data Warehouse.

Data Warehouses store data in a hierarchical format using files and folders. This is not the case with a Data Lake as Data Lake Architecture is a flat architecture. In a Data Lake, every data element is identified by a unique identifier and a set of metadata information.

Goals

> Building and maintaining a Data Lake have five main goals: unifying the data, full query access, performance and scalability, progression, and costs.

Unification: Data Lake is a perfect solution to accumulate all the data from distinct data sources (ERP, CRM, logs, data partners data, internal generated data) in one place. The Data Lake Architecture makes it easier for companies to get a holistic view of data and generate insights from it.

Full Query Access: storing data in Data Lakes allows full access to data that can be directly used by BI tools to pull data whenever needed. ELT process is a flexible, reliable, and fast way to load data into Data Lake and then use it with other tools.

Performance and Scalability: Data Lake Architecture supports fast query processing. It enables users to perform ad hoc analytical queries independent of the production environment. Data Lake provides faster querying and makes it easier to scale up and down. Data Lake offer business agility.

Progression: getting data in one place is a necessary step before progressing to other stages because loading data from one source makes it easier to work with BI tools. Data Lake helps you make data cleaner and error-free data that has less repetition.

Costs: S3 repositories are a cost-efficient storage of large volumes of data.

Data Lake Architecture

Data Lake Architecture Framework.

> Data Lakes are often structured in zones or layers models. These models define in which processing degrees (raw, cleansed, aggregated) data are available in the data lake, and how they are governed (regarding access rights, data quality, and responsibilities).

Zones are similar to the layers in data warehousing, but data may not move through all zones or even move back.

Layers

Data Lake Layers.

The following data lake model approach is structured in layers:

- Ingestion Layer
- Distilation Layer
- Processing Layer
- Insights Layer
- Unified Operations Layer

The **Raw Data** entering the Data Lake consists of the organizations internal data (Operational Systems), specially relational data from databases, also streaming and batch data from data partners.

In the other extreme, representing the data leaving the Data Lake, the **Business Systems**, consists of databases, the Data Warehouse, dashboards, reports, and external data connections.

The first three layers constitute the medallion architecture, which is a is a data design pattern used to logically organize data in a Data Lake (similar as in a Data Lakehouse), with the goal of incrementally and progressively improving the structure and quality of data as it flows through each layer of the architecture (from Bronze ⇒ Silver ⇒ Gold layer tables). Medallion architectures are sometimes also referred to as "multi-hop" architectures.

Ingestion Layer (Bronze)

> The purpose of the Ingestion Layer of the Data Lake Architecture is to ingest raw data into the Data Lake. There is no data modification in this layer. This is where we land all the data from external source systems.

The table structures in this layer correspond to the source system table structures "as-is," along with any additional metadata columns that capture the load date/time, process ID, etc. The focus in this layer is quick Change Data Capture and the ability to provide an historical archive of source (cold storage), data lineage, auditability, reprocessing if needed without rereading the data from the source system.

The layer can ingest raw data in real-time or in batches, which is in turn organized into a logical folder structure. The Ingestion Layer can pull data from different external sources, like social media platforms.

Distillation Layer (Silver)

> The purpose of the Distillation Layer of the Data Lake Architecture is to convert the data stored in the Ingestion (Bronze) Layer in a Structured format for analytics.

The data is matched, denormalized, merged, conformed, cleansed, and derive "just-enough" so that the Silver layer can provide an "Enterprise view" of all its key business entities, concepts and transactions (for example, master customers, stores, non-duplicated transactions and cross-reference tables). The data in this layer becomes uniform in terms of format, encoding, and data type (parquet).

The Silver layer brings the data from different sources into an Enterprise view and enables self-service analytics for ad-hoc reporting, advanced analytics and ML. It serves as a source for Departmental Analysts, Data Engineers and Data Scientists to further create projects and analysis to answer business problems via enterprise and departmental data projects in the Gold Layer.

In the lakehouse data engineering paradigm (of which we're extending to the Data Lake), typically the ELT methodology is followed vs. ETL - which means only minimal or "just-enough" transformations and data cleansing rules are applied while loading the Silver layer. Speed and agility to ingest and deliver the data in the data lake is prioritized, and a lot of project-specific complex transformations and business rules are applied while loading the data from the Silver to Gold layer. From a data modeling perspective, the Silver Layer has more 3rd-Normal Form like data models. Data Vault-like, write-performant data models can be used in this layer.

Processing Layer (Gold)

> This layer of the Data Lake Architecture executes user queries and advanced analytical tools on the Structured Data.

The processes can be run in batch, in real-time, or interactively. It is the layer that implements the business logic and analytical applications consume the data. It is also known as the Trusted, Gold, or Production-Ready Layer.

It is typically organized in consumption-ready "project-specific" databases. The Gold layer is for reporting and uses more de-normalized and read-optimized data models with fewer joins. The final layer of data transformations and data quality rules are applied here. Final presentation layer of projects such as Customer Analytics, Product Quality Analytics, Inventory Analytics, Customer Segmentation, Product Recommendations, Marking/Sales Analytics etc. fit in this layer. We see a lot of Kimball style star schema-based data models or Inmon style Data marts fit.

Often, the Data Marts (and Data Warehouse data) from the traditional RDBMS technology stack are ingested into the Gold layer.

Insights Layer

> This layer of the Data Lake Architecture acts as the query interface, or the output interface, of the Data Lake.

It uses SQL and NoSQL queries to request or fetch data from the Data Lake. The queries are normally executed by company users who need access to the data. Once the data is fetched from the Data Lake, it is the same layer that displays it to the user for viewing.

Some examples include Amazon QuickSight, an AWS native BI tool and allows users to connect with software-as-a-service (SaaS) applications such as Salesforce or ServiceNow, third-party databases such as MySQL, Postgres, and SQL Server, as well as native AWS services including Amazon Athena, an

interactive query service that allows them to analyze unstructured data in Amazon S3 data lakes using standard SQL queries. While QuickSight doesn't connect directly to the data lake, integration with Amazon Athena allows BI users to query data inside the lake without having to move data or build an ETL pipeline.

Unified Operations Layer

> This layer governs system management and monitoring.

It includes auditing and proficiency management, data management, workflow management. AWS data lake environments and monitoring tools and best practices are described here.

Zones

Zone Reference Model for Enterprise-Grade Data Lake Management - Data Flow.

The following data lake approach is explored by the University of Stuttgart and Bosch GmbH, and known as Zone Reference Model for Enterprise-Grade Data Lake Management. It consists of:

- Landing Zone
- Raw Zone

- Harmonized Zone
- Distilled Zone
- Delivery Zone
- Explorative Zone

PlantUML rendering error: Failed to render inline diagram (Failed to generate PlantUML diagrams, PlantUML exited with code 127 (sh: 1: plantuml.exe: not found).).

Zone Reference Model for Enterprise-Grade Data Lake Management - Meta-model for zones - Attributes.

The following describes how a zone interacts with other zones and the outside world.

PlantUML rendering error: Failed to render inline diagram (Failed to generate PlantUML diagrams, PlantUML exited with code 127 (sh: 1: plantuml.exe: not found).).

Zone Reference Model for Enterprise-Grade Data Lake Management - Meta-model for zones - Interactions.

All zones contain a protected part. This part is encrypted and secured, and stores data that need extensive protection (for example, PII, personal data). Data wander from the protected part of one zone to the protected part of the next zone. They may only leave the protected part after being desensitized (for example, by anonymization). Data in this part are subject to strict access controls and governance. The protected part shares all other characteristics with the rest of the zone it is in.

Landing Zone

> The Landing Zone is the first zone of the data lake. Data are ingested as batch or as data stream from the sources.

The Landing Zone is beneficial when the requirements of the ingested data and those of the Raw Zone diverge. For example, data might need to be

ingested at a vast rate due to its volume and velocity. If the technical implementation of the Raw Zone cannot provide this high ingestion rate, a Landing Zone can function as a mediator in between: data are ingested at a high rate into the Landing Zone, and then are forwarded to the Raw Zone as batches. Examples of data coming to the Landing zone include data streamed by Kafka, Amazon Kinesis, Amazon SQS, RabbitMQ, Apache Spark, etc.

For the data characteristics, data ingested into the Landing Zone remains mostly raw. Their granularity remains raw, just like in the source systems. The schema of the data is not changed; they can simply be copied in their source system format. However, their syntax might be changed. Basic transformations are allowed upon ingestion into the Landing Zone, such as adjusting the character set of strings or transforming timestamps into a common format. In addition, data may be masked or anonymized to comply with legal regulations. Aside from these changes, the semantic of the data remains the same as in the source systems.

Raw Zone

> All data in the data lake is available in mostly raw format in the Raw Zone. Only basic transformations (see Landing Zone) are applied on the data. If the Landing Zone is omitted, these transformations are performed in the Raw Zone.

Differently from the Landing Zone, the Raw Zone stores data persistently. In general, data should neither be manipulated nor deleted from the Raw Zone. This zone persists (when possible) the original data type (json, csv, xml).

Harmonized Zone

> The Harmonized Zone is the place where master data are accessible for analyses.

A subset of the data stored in the Raw Zone is passed to the Harmonized Zone in a demand-based manner. It is important to note that these data are not deleted from the Raw Zone. Instead, the Harmonized Zone contains a copy of or a view on the data in the Raw Zone. The Harmonized Zone is also the place where master data are accessible for analyses. As these data are crucial for enterprises, master data management is of high importance in the data lake. Thus, they should exclusively be accessed after being cleansed.

The data characteristics in this zone differ greatly from those in the Raw Zone. Data schema and syntax change when compared to the source data. Data from different source systems are integrated into a consolidated schema, regardless of their structure. The data syntax is also consolidated in the Harmonized Zone: when data from multiple source systems are merged, data types have to be adapted.

The aim of the Harmonized Zone is to provide a harmonized and consolidated view on data. To this end, the Harmonized Zone uses a standardized modeling approach (dimensional modeling or Data Vault) that all of the enterprise's data are modeled in. The files in this zone facilitates the ingestion (parquet).

Distilled Zone

Prepares data for processing and facilitates ingestion.

In contrast to the Raw and Harmonized Zone, where the focus is to quickly make data available for use, the Distilled Zone focuses on increasing the efficiency of following analyses by preparing the data accordingly. The granularity of the data may be changed (for example, data may be aggregated for the calculation of KPIs). Complex processing is applied that change the data's semantics but are too extensive for the Landing Zone, Raw Zone, and Harmonized Zone. However, the schema might also change slightly, depending on the supported use case (for example, fields to enrich the data could be added). The files in this zone facilitates the ingestion (parquet).

Explorative Zone

> The Explorative Zone is the place where data scientists can play with and flexibly use the data.

Data scientists can use and explore data in the data lake in any way they desire, except for sensitive data. These data are only usable according to strict rules. Granularity, schema, syntax, and semantic may be changed in any way necessary for analyses.

Delivery Zone

> In the Delivery Zone, small subsets of data are tailored to specific usage and applications.

This does not only include analytical use cases, such as reporting and OLAP, but also operational use cases. This zone thus provides functionality similar to data marts and operational data stores in data warehousing. Data from this zone may be forwarded to external data sinks.

The Delivery Zone especially supports users with little knowledge on data analytics. Data have to be easily findable and importable into various analytics tools. As for the modeling approach, data are available in whatever format supports the intended use case best, for example, dimensional modeling for OLAP, or flat tables for operational use.

Zones Comparisom

| | **Landing** | **Raw** | **Harmonized** | **Distilled** | **Explorative** | **Delivery** | |---: |:---: |:---: |:---: |:---: |:---: |:---: | | **Granularity** | Raw | Raw | Raw | Aggregated | Any | Any | | **Schema** | Any | Any | Consolidated | Consolidated, Enriched | Any | Any | | **Syntax** | Basic transformations | Basic transformations | Consolidated | Consolidated | Any | Any | | **Semantics** | Mostly unchanged* | Mostly unchanged* | Mostly unchanged* | Complex processing | Any | Any

| | **Governed** | True | True | True | True | False | True | | **Historized** | False | True | True | True | N/A | N/A | | **Persistent** | False | True | True | True | False | True | | **Has Protected Part** | True | True | True | True | True | True | | **Use Case Dependent** | False | False | False | True | True | True | | **User Groups** | Systems, Processes | Data Scientists, Systems, Processes | Data Scientists, Systems, Processes | Data Scientists, Domain Experts, Systems, Processes | Data Scientists | Anyone | | **Modelling Approach** | Any | Any | Standardized | Standardized | Any | Any |

Zone Reference Model for Enterprise-Grade Data Lake Management - Zones comparison.

Sandbox

Also known as the Analytics Sandbox, it provides data scientists and advanced analysts with a place for data exploration.

An Analytics Sandbox is a separate environment that is part of the overall data lake architecture, meaning that it is a centralized environment meant to be used by multiple users and is maintained with the support of IT. Some key characteristics of this layer:

- The environment is controlled by the analyst
- Allows them to install and use the data tools of their choice
- Allows them to manage the scheduling and processing of the data assets
- Enables analysts to explore and experiment with internal and external data
- Can hold and process large amounts of data efficiently from many different data sources; big data (unstructured), transactional data (structured), web data, social media data, documents, etc.

There are many advantages to having an Analytics Sandbox as part of your data architecture. The most important is that it decreases the amount of time that it takes a business to gain knowledge and insight from their data. It does this by providing an on-demand/always ready environment that allows

analysts to quickly dive into and process large amounts of data and prototype their solutions without kicking off a big BI project. In other words, it enables agile BI by empowering your advanced users.

Another major benefit to the business and IT team is that by giving the business a place to prototype their data solutions it allows the business to figure what they want on their own without involving IT. When they decide that a solution is adding business value, it becomes a good candidate for something that should be productionized and built into the Data Warehouse process at some point. This saves both teams a lot of time and effort.

Maturity Stages

The implementation of a Data Lake solution consists of some main maturity stages.

1. Handle and ingest data at scale
2. Building the analytical muscle
3. Data Warehouse and Data Lake working in unison
4. Enterprise capability

Handle and Ingest data at scale

> This stage consists in improving the ability to transform and analyze data.

Building the analytical muscle

> This stage involves improving the ability to transform and analyze data.

In this stage, the company start acquiring more data and building applications. In this stage, capabilities of the Data Warehouse and the Data Lake are used together.

Data Warehouse and Data Lake work in unison

> This step involves getting data and analytics into the hands of as many people as possible.

In this stage, the Data Lake and the Data Warehouse start to work in a union. Both playing their part in analytics.

Enterprise capability in the lake

> In this maturity stage of the data lake, enterprise capabilities are added to the Data Lake.

It includes the adoption of information governance, information lifecycle management capabilities, and Metadata management.

Key Components of Data Lake Architecture

#TODO: draw and include here diagram of data lake architecture including data sources, data processing layer, and data targets.

Data Ingestion

> Data Ingestion allows connectors to get data from a different data sources and load into the Data Lake.

Data Ingestion supports:

- All types of Structured, Semi-Structured, and Unstructured data.
- Multiple ingestions like Batch, Real-Time, One-time load.
- Many types of data sources like Databases, Webservers, Emails, and FTP.

Data Storage

Data storage should be scalable, offers cost-effective storage and allow fast access to data exploration. It should support various data formats.

Data Governance

Data governance is a process of managing availability, usability, security, and integrity of data used in an organization.

Security

Security needs to be implemented in every layer of the Data Lake. It starts with Storage, Unearthing, and Consumption. The basic need is to stop access for unauthorized users. It should support different tools to access data with easy to navigate GUI and Dashboards.

Authentication, Accounting, Authorization and Data Protection are some important features of Data Lake security.

Data Quality

Data quality is an essential component of Data Lake architecture. Data is used to exact business value. Extracting insights from poor quality data will lead to poor quality insights.

Data Discovery

Data Discovery is another important stage before you can begin preparing data or analysis. In this stage, tagging technique is used to express the data understanding, by organizing and interpreting the data ingested in the Data Lake.

Data Auditing

Data auditing helps to evaluate risk and compliance.

The main Data auditing tasks are:

- Tracking changes to important dataset elements
- Captures how/when/who changes to these elements.

Data Lineage

This component deals with data's origins. It mainly deals with where it movers over time and what happens to it. It eases errors corrections in a data analytics process from origin to destination.

Data Exploration

It is the beginning stage of data analysis. It helps to identify right dataset is vital before starting Data Exploration.

All given components need to work together to play an important part in Data Lake building easily evolve and explore the environment.

A Data Warehouse (DWH), also known as Enterprise Data Warehouse (EDW) is a central repository of information that can be analyzed to make more informed decisions.

Data flows into a data warehouse from Data Lake, transactional systems, relational databases, and other sources, typically on a regular cadence. Business analysts, data engineers, data scientists, and decision makers access the data through Business Intelligence (BI) tools, SQL clients, and other analytics applications.

Goals Data Warehouse Architecture Data Sources Warehouse Metadata Summarized data End-User access Tools Two-Tier Architecture Three-Tier Architecture Reconciled Layer Middle Tier AWS Redshift & Snowflake Databricks Lakehouse Platform Data Lakehouse vs Data Warehouse vs Data Lake OLAP Servers Relational OLAP Servers (ROLAP) Multidimensional OLAP Servers (MOLAP) Hybrid OLAP Servers (HOLAP) OLAP Servers options Data Modeling Methodologies Kimball (Bottom-Up) Advantages Disadvantages Inmon (Top-Down) Advantages Disadvantages Hybrid Vault Bus Architecture Maturity Stages Key Components of a Data Warehouse References Goals When implementing a data warehouse, the main goals are to achieve:

Consistency: maintain a uniform format to all collected data, making it easier for corporate decision-makers to analyze and share data insights with their colleagues. Standardizing data from different sources also reduces the risk of error in interpretation and improves overall accuracy.

Decision-making: successful business leaders develop data-driven strategies and rarely make decisions without consulting the facts. Data warehousing improves the speed and efficiency of accessing different data sets and makes it easier for corporate decision-makers to derive insights that will guide the business and marketing strategies that set them apart from their competitors.

Improving: allow business leaders to quickly access the organization historical activities and evaluate initiatives that have been successful — or unsuccessful — in the past. This allows executives to see where they can adjust their strategy to decrease costs, maximize efficiency and increase business results.

Single Source of Truth: the whole company would benefit on having a single source of truth, specially when there are multiple data sources to a common business dimension.

Data Warehouse Architecture

There are several data warehouses architecture approaches available. Data warehouses would have in common some key components:

Data Sources In most architectures approaches, it's the Source Layer, or Data Source Layer, and consists on all the data sources the Warehouse Layer will consume.

Operational System: is a method used in data warehousing to refer to a system that is used to process the day-to-day transactions of an organization.

In our case, that's the data coming from our internal services (user service, commerce service, etc.). It'll be available in the Data Lake, in parquet format.

Flat Files System: is a system of files in which transactional data is stored, and every file in the system must have a different name.

In the current DWH implementation, that's the data coming from external providers as Braze and Segment. I'll be also available in the Data Lake, in parquet format.

Warehouse In most architectures approaches, it's the Warehouse Layer, or Data Warehouse Layer, and consists on all the data stored in RDBMS database with available gateway access (ODBC, JDBC, etc.). It also contains the metadata, and some degree of data summarization, and business logic applied, which differentiate an DWH database from a Production database.

Metadata Metadata is the road-map to a data warehouse, it defines the warehouse objects, and acts as a directory. This directory helps the decision support system to locate the contents of a data warehouse.

In the current DWH implementation, there's no metadata management tool yet in use. The only metadata we're applying is when the data was last updated in the Data Lake (dl_updated_at), when the data first entered the Data Warehouse (dwh_created_at), and when it was last updated (dwh_updated_at).

It should soon be extended to:

A description of the Data Warehouse structure, including the warehouse schema, dimensions, hierarchies, data mart locations, contents, etc.

Operational metadata, which usually describes the currency level of the stored data (for example, active, archived or purged), and warehouse monitoring information (for example, usage statistics, error reports, audit, etc).

System performance data, which includes indices, used to improve data access and retrieval performance.

Information about the mapping from operational databases, which provides source RDBMSs and their contents, cleaning and transformation rules, etc.

Summarization algorithms, predefined queries, and reports business data, which include business terms and definitions, ownership information, etc.

Metadata management tool example.

Summarized data The area of the data warehouse that maintains all the predefined lightly and highly summarized (aggregated) data. The main goal is to speed up query performance, and the summarized records are updated continuously as new information is loaded into the warehouse.

In the current DWH implementation, the data is being summarized directly in the Data Marts, using dbt (bottom-down approach). That means we do not have summarized data stored and available for BI teams to work with, nor have a Single Source of Truth applied at data warehouse layer.

End-User access Tools The principal purpose of a data warehouse is to provide information to the business for strategic decision-making. These end-users interact with the warehouse using end-client access tools.

The examples of some of the end-user access tools can be:

Reporting and Query Tools

Application Development Tools

Executive Information Systems Tools

Online Analytical Processing Tools

Data Mining Tools

In our case, the only tool currently consuming the Data Marts, is Tableau.

Two-Tier Architecture The current architecture of our data warehouse is close to a Two-Tier architecture. To reach a proper two-tier architecture, we still have to adopt a Data Staging solution, and a Metadata Management solution. The last great step to fully implement it, is to define how the data can be accessed by external tools.

A data staging solution in this context means to have data cleansed and available with a standard schema at the data warehouse level, before the data enters the data marts pipelines.

Once we fully migrate to this architecture (as a middle step to reach a Three-Tier architecture), our architecture will be as follows:

Data Source Layer: A data warehouse system uses a heterogeneous source of data. That data is stored initially to corporate relational databases or legacy databases, or it may come from an information system outside the corporate walls.

Data Staging Layer: The data stored to the source should be extracted, cleansed to remove inconsistencies and fill gaps, and integrated to merge heterogeneous sources into one standard schema. The ETLs can combine heterogeneous schemata, extract, transform, cleanse, validate, filter, and load source data into a data warehouse. Note that this can be achieved in two ways:

Having the Distillation Layer (Silver) in the Data Lake as the Data Staging Layer.

Creating a separated database within the Data Warehouse, or a separated database schema. Following the principle that all the data in the data warehouse should be cleaned and have high quality standards, a separated database should be preferred.

Data Warehouse Layer: Information is saved to one logically centralized individual repository: a data warehouse. The data warehouses can be directly

accessed, but it can also be used as a source for creating data marts, which partially replicate data warehouse contents and are designed for specific enterprise departments. Metadata repositories store information on sources, access procedures, data staging, users, data mart schema, and so on.

Analysis Layer: In this layer, integrated data is efficiently, and flexibly accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios. It should feature aggregated information navigators, complex query optimizers, and customer-friendly GUIs.

Three-Tier Architecture Reconciled Layer Once we fully implement a Two-Tier Architecture, the next step is to adopt a Three-Tier Architecture. For it it's necessary to implement a Reconciled Layer, and a Middle Tier.

Once we fully implement the Reconciled Layer, our architecture will be as follows:

The Reconciled Layer sits between the source data and data warehouse. The main advantage of the reconciled layer is that it creates a standard reference data model for the whole company. At the same time, it separates the problems of source data extraction and integration from those of data warehouse population. In some cases, the reconciled layer is also directly used to accomplish better operational tasks, such as producing daily reports that cannot be satisfactorily prepared using the corporate applications or generating data flows to feed external processes periodically to benefit from cleaning and integration.

This architecture is especially useful for the extensive, enterprise-wide systems. A disadvantage of this structure is the extra file storage space used through the extra redundant reconciled layer. It also makes the analytical tools a little further away from being real-time.

Middle Tier As the name of the architecture suggests, it consists of three tiers (levels):

Bottom Tier (Data): data warehouse server with functional gateway (ODBC, JDBC, etc.).

Middle Tier (Application): houses the business logic used to process user inputs. Example: OLAP Servers, Snowflake, Apache Redshift, Databricks Data Lakehouse Platform.

Top Tier (Presentation): front-end tools.

The Bottom and Top tiers were already discussed in details in the previous steps, so we only have left the implementation of the Middle Tier, very important to enable fast querying of the data warehouse. It is important to note the solutions to the middle tier are often referred as the Data Warehouse itself, but they're only the Application tier/level in a complete data warehouse solution (It's like saying Redshift is the DWH, when it's just one part of the complete DWH solution).

Once we fully implement the Middle Tier, our architecture will be as follows:

One could argue that OLAP is dead, at least in the traditional format (solutions like Mondrian), and cloud/modern solutions should be applied to accompany business fast demand for data. In this sense, given costs restrictions, solutions like Apache Kylin seem to be a better approach (details below). When costs are less restricted, and/or data demands increase, other solutions like AWS Redshift, Snowflake, and Databricks Lakehouse are preferred/recommended.

AWS Redshift & Snowflake AWS Redshift and Snowflake comparison available in this miro dashboard (to be migrated to Confluence soon).

Databricks Lakehouse Platform It combines the ACID transactions and data governance of data warehouses with the flexibility and cost-efficiency of data lakes to enable business intelligence (BI) and machine learning (ML) on all data.

Data Lakehouse vs Data Warehouse vs Data Lake Data in the data warehouse is easy to use, but harder to store. The opposite is true for the data lake: it's easy to ingest and store data, but a pain to consume and query.

The data lakehouse has a layer design, with a warehouse layer on top of a data lake. This architecture, which enables combining structured and unstructured data, makes it efficient for business intelligence and business analysis. Data lakehouses provide structured storage for some types of data and unstructured storage for others while keeping all data in one place.

OLAP Servers Relational OLAP Servers (ROLAP) They use a relational or extended-relational DBMS to save and handle warehouse data, and OLAP middleware to provide missing pieces. They work primarily from the data that resides in a relational database, where the base data and dimension tables are stored as relational tables. This model permits the multidimensional analysis of data.

This technique relies on manipulating the data stored in the relational database to give the presence of traditional OLAP's slicing and dicing functionality.

Advantages Can handle large amounts of information: the data size limitation of ROLAP technology is depends on the data size of the underlying RDBMS. So, ROLAP itself does not restrict the data amount.

RDBMS already comes with a lot of features. So ROLAP technologies, (works on top of the RDBMS) can control these functionalities.

Disadvantages Performance can be slow: each ROLAP report is a SQL query (or multiple SQL queries) in the relational database, the query time can be prolonged if the underlying data size is large.

Limited by SQL functionalities: ROLAP technology relies on upon developing SQL statements to query the relational database, and SQL statements do not suit all needs.

Multidimensional OLAP Servers (MOLAP) It is based on a native logical model that directly supports multidimensional data and operations. Data are stored physically into multidimensional arrays, and positional techniques are used to access them.

One of the significant distinctions of MOLAP against a ROLAP is that data are summarized and are stored in an optimized format in a multidimensional cube, instead of in a relational database. In MOLAP model, data are structured into proprietary formats by client's reporting requirements with the calculations pre-generated on the cubes.

Advantages Excellent Performance: a MOLAP cube is built for fast information retrieval, and is optimal for slicing and dicing operations.

Can perform complex calculations: all evaluation have been pre-generated when the cube is created. Hence, complex calculations are not only possible, but they return quickly.

Disadvantages Limited in the amount of information it can handle: Because all calculations are performed when the cube is built, it is not possible to contain a large amount of data in the cube itself.

Hybrid OLAP Servers (HOLAP) It incorporates the best features of MOLAP and ROLAP into a single architecture. It saves more substantial quantities of detailed data in the relational tables while the aggregations are stored in the pre-calculated cubes. HOLAP also can drill through from the cube down to the relational tables for delineated data.

Advantages It provides benefits of both MOLAP and ROLAP.

It provides fast access at all levels of aggregation.

It balances the disk space requirement, as it only stores the aggregate information on the OLAP server and the detail record remains in the relational database. So no duplicate copy of the detail record is maintained.

Disadvantages HOLAP architecture is very complicated because it supports both MOLAP and ROLAP servers.

OLAP Servers options Apache Kylin Only supports MOLAP and Offline data storage modes. It supports both SQL and MDX queries, have RESTful API capabilities (also ODBC, and JDBC), and can be integrated/connect with Tableau (also Redash, Superset, Zeppelin, Qlik, and Excel).

It supports Real Time processing, partitioning, usage based optimizations, load balancing and clustering. It supports LDAP, SAML, Kerboros authentication.

Mondrian OLAP Server Only supports ROLAP data storage modes. It supports MDX queries but not SQL, and have REST API capabilities. Does not natively connect with Tableau, but queries can be performed via Java APIs. It supports Real Time processing, and partitioning.

Data Modeling Methodologies Being one of the most important topics of data warehouse design and architecture, the data modeling methodology choosing process is arduous and polemic, and will impact the whole design and implementation of the data warehouse solution.

Though not planned, or discussed, the current architecture of the data warehouse, is of type Kimball (bottom-up), which means the data marts are first formed based on the business requirements. There are lots of advantages and disadvantages of priming this approach over a top-down approach (or any of the hybrid or alternative methodologies).

Kimball (Bottom-Up) The design of the Data Marts comes from the business requirements. The primary data sources are then evaluated, and an Extract, Transform and Load (ETL) tool is used to fetch data from several sources and load it into a staging area of the relational database server. Once data is uploaded in the data warehouse staging area, the next phase includes loading data into a dimensional data warehouse model that is denormalized by nature. This model partitions data into the fact and dimension tables. Kimball dimensional modeling allows users to construct several star schemas to fulfill various reporting needs.

Kimball Approach to Data Warehouse Lifecycle. Advantages Fast to construct (quick initial phase): it is fast to construct as no normalization is involved, which means swift execution of the initial phase of the data warehousing design process.

Simplified queries: in a star schema, most data operators can easily comprehend it because of its denormalized structure, which simplifies querying and analysis.

Simplified business management: the data warehouse system footprint is trivial because it focuses on individual business areas and processes rather than the whole company. So, it takes less space in the database, simplifying system management.

Fast data retrieval: as data is segregated into fact tables and dimensions.

Smaller teams: a smaller team of designers and planners is sufficient for data warehouse management because data source systems are stable, and the

data warehouse is process-oriented. Also, query optimization is straightforward, predictable, and controllable.

Deeper insights: it allows business intelligence tools to deeper across several star schemas and generates reliable insights.

Disadvantages No Single Source of Truth: Data isn't entirely integrated before reporting, so the idea of a single source of truth is lost.

Too prone to data irregularities: this is because in denormalization technique, redundant data is added to database tables.

Too difficult and expensive to add new columns: performance issues may occur due to the addition of columns in the fact table, as these tables are quite in-depth. The addition of new columns can expand the fact table dimensions, affecting its performance.

Can't respond (well) to business changes: it is too difficult to alter the models.

Not BI-friendly: as it is business process-oriented, instead of focusing on the company as a whole, it cannot handle all the BI reporting requirements.

Inconsistent dimensional view: this model is not strong as top-down approach as dimensional view of data marts is not consistent as it is in Inmon approach.

In brief, the Kimball approach have a low start-up cost, is faster to deliver the first phase of the data warehouse design, is faster to release to production (first version), but is suitable for Tactical business decision support requirements (versus Strategic), and addresses individual business requirements (vs Enterprise-wide). Another important topic that derives from this methodology approach is the Data Warehouse Bus Architecture (described in next topics).

Inmon (Top-Down) On the other hand, Bill Inmon, the father of data warehousing, came up with the concept to develop a data warehouse which identifies the main subject areas and entities the enterprise works with, such as customers, product, vendor, etc. Inmon's definition of a data warehouse is that it is a "subject-oriented, nonvolatile, integrated, time-variant collection of data in support of management's decisions".

The model then creates a thorough, logical model for every primary entity. For instance, a logical model is constructed for products with all the attributes associated with that entity. This logical model could include many entities, including all the details, aspects, relationships, dependencies, and affiliations.

The Inmon design approach uses the normalized form for building entity structure, avoiding data redundancy as much as possible. This results in clearly identifying business requirements and preventing any data update irregularities. Moreover, the advantage of this top-down approach in database design is that it is robust to business changes and contains a dimensional perspective of data across data mart.

Next, the physical model is constructed, which follows the normalized structure. This Inmon model creates a Single Source of Truth for the whole business to consume. Data loading becomes less complex due to the normalized structure of the model. However, using this arrangement for querying is challenging as it includes numerous tables and links.

This Inmon data warehouse methodology proposes constructing data marts separately for each division, such as finance, marketing sales, etc. All the data entering the data warehouse is integrated. The data warehouse acts as a single data source for various data marts to ensure integrity and consistency across the enterprise.

Advantages Single Source of Truth: the data warehouse acts as a unified source of truth for the entire business, where all data is integrated.

Very low data redundancy: there's less possibility of data update irregularities, making the data warehouse ETL processes more straightforward and less susceptible to failure.

Great flexibility: it's easier to update the data warehouse in case there's any change in the business requirements or source data.

BI-friendly: It can handle diverse company-wide reporting requirements.

Disadvantages Increasing complexity: it increases as multiple tables are added to the data model with time.

Skilled Human Resources: resources skilled in data warehouse data modeling are required, which can be expensive and challenging to find.

Slow setup: the preliminary setup and delivery are time-consuming.

Expert management: this approach requires experts to manage a data warehouse effectively.

In brief, the Inmon have a high start-up cost, requires more time to be in production and meet business needs (very large projects with a very broad scope), and requires a bigger team os specialists, but is more suitable for for systems and business changes, better integrates with the whole company, favors Strategic business decision support requirements (vs Tactical), and facilitates Business Intelligence development.

Hybrid In a hybrid model, the data warehouse is built using the Inmon model, and on top of the integrated data warehouse, the business process oriented data marts are built using the star schema for reporting.

The hybrid approach provides a Single Source of Truth for the data marts, creating a highly flexible solutions from a BI point of view.

Based on the Hub and Spoke Architecture, the hybrid design methodology can also make use of Operational Data Stores (ODS), integrating and cleaning data from multiple data sources. The information is then parsed into the actual Data Warehouse.

Hybrid methods will normally keep the data in the 3rd normal form, reducing redundancy. Although normal relational database is not efficient for BI reports. Data marts for specific reports can then be built on top of the data warehouse solution.

When the data is denormalized, all the data available is pulled (as advocated by Inmon) while using a denormalized design (as advocated by Kimball). One example is the Carry Forward method.

Another hybrid methodology is the Data Vault, discussed below.

Example of a Hybrid Methodology approach. Vault The Vault Data Modeling is a hybrid design, consisting of the best of breed practices from both 3rd

normal form and star-schema.

It is not a true 3rd normal form, and breaks some of the rules that 3NF dictates. It is a top-down architecture with bottom-up design, geared to be strictly a data warehouse. It is not geared to be end-user accessible, which when built, still requires the user of a data mart or star-schema based release are for business purposes.

Data Vault data modeling breaks data into a small number of standard components – the most common of which are Hubs, Links and Satellites.

Hubs are entities of interest to the business. They contain just a distinct list of business keys and metadata about when each key was first loaded and from where.

Links connect Hubs and may record a transaction, composition, or other type of relationship between hubs. They contain details of the hubs involved (as foreign keys) and metadata about when the link was first loaded and from where.

Satellites connect to Hubs or Links. They are Point in Time: so we can ask and answer the question, “what did we know when?”. Satellites contain data about their parent Hub or Link and metadata about when the data was loaded, from where, and a business effectivity date.

The data model of the data warehouse is constructed using these components. These are:

Standard: each component is always constructed the same way.

Simple: easy to understand, and with a little practice, easy to apply them to model your system.

Connected: hubs only connect to links and satellites, links only connect to hubs and satellites, and satellites only connect to hubs or links.

Data Vault has staging, vault and mart layers. Star schemas live in the mart layer, each star schema exposes a subset of the vault for a particular group of users. Typically, hubs and their satellites form dimensions, links and their satellites form facts.

A Data Vault complements the Data Lake and is a solution for organizations that need to integrate and add structure to the data held in the Data Lake.

Data Vault Modeling Methodology. This is how our Data Warehouse Architecture would look like once the Data Vault Modeling Methodology is implemented:

Bus Architecture … Inflow, Upflow, Downflow, Outflow and Meta flow.

Maturity Stages …

Key Components of a Data Warehouse Data Ingestion: allows connectors to get data from a different data sources and load into the Data Warehouse. The data will normally come from the Data Lake and External Sources connection (Fivetran), through multiple ETLs (Airflow, services, apps, ETL tools and platforms, etc.).

Data Storage: the data is stored in the data warehouse database, a relational database (RDBMS), like Postgres.

Data Governance: is a process of managing availability, usability, security, and integrity of data used in an organization.

Security: it needs to be implemented in every layer of the Data Warehouse. It includes setting up the data warehouse read-only by default, and setting up custom User Groups. It also includes the access to the databases (VPCs, VPNs, Whitelisting, etc.), strong and active DevOps monitoring and the enforcing of best practices in all levels of the data warehouse environment (data ingestion, data marts consumption, ETLs design, etc.).

Data Quality: it is an essential component of Data Warehouse architecture. Data is used to exact business value. Extracting insights from poor quality data will lead to poor quality insights.

Data Discovery: it is another important stage before you can begin preparing data or analysis. All this rely on good metadata, and data modeling.

Data Auditing: it helps to evaluate risk and compliance. Two major Data auditing tasks are tracking changes to the key dataset.

Tracking changes to important dataset elements.

Captures how/when/who changes to these elements.

Data Lineage: it deals with data's origins. It mainly deals with where it movers over time and what happens to it. It eases errors corrections in a data analytics process from origin to destination. Some data modeling techniques may facilitate lineage in comparison to others (Vault vs Kimball vs Inmon).

Data Exploration: it is the beginning stage of data analysis. It helps to identify right dataset is vital before starting Data Exploration. All given components need to work together to play an important part in Data Warehouse building easily evolve and explore the environment.

References Data Warehouse - AWS

Data Warehouse Architecture - javatpoint Data Warehouse Architecture with Introduction, What is Data Warehouse, History of Data Warehouse, Data Warehouse Components, Operational Database Vs Data Warehouse etc.

www.javatpoint.com

Data Warehouse Architecture, Components & Diagram Concepts This data warehouse architecture tutorial covers all the basic to advance stuff like definitions, characteristics, architectures, components, data marts, and more.

Guru99

Data Warehouse Architecture - GeeksforGeeks A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

GeeksforGeeks

Data Warehouse Architecture: Traditional vs. Cloud Data warehouse architecture is changing. Learn about traditional EDW vs. cloud-based architectures with lower upfront cost, improved scalability and performance.

Panoply

Kimball vs. Inmon in Data Warehouse Architecture

We will discuss about the Kimball vs. Inmon in data warehouse architecture and design approach. We also answer the question of how to choose Kimball or Inmon's architecture to build data warehouse.

zentut Data Warehouse Concepts: Kimball vs. Inmon Approach

Inmon vs Kimball: Which data warehouse concept should you use to design a data warehouse. Find out in this blog.

Astera

Data vault modeling

Data vault modeling is a database modeling method that is designed to provide long-term historical storage of data coming in from multiple operational systems. It is also a method of looking at historical data that deals with issues such as auditing, tracing of data, loading speed and resilience to change as well as emphasizing the need to trace where all the data in the database came from. This means that every row in a data vault must be accompanied by record source and load date attributes, enabling an auditor to trace values back to the source. It was developed by Daniel (Dan) Linstedt in 2000.

Wikipedia

Data Warehousing concepts: Kimball vs. Inmon vs. Hybrid vs. Vault Knowledge sharing with intelligence beyond rational...

Drazda

Data warehouse 12 reconciled data layers

Reconciled Data Layers during the ETL process

SlideShare

Full screen view https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c80f8aaea5bf58846b0125b460401fed8230c2d2

Data Warehousing - Metadata Concepts Data Warehousing Metadata Concepts - Metadata is simply defined as data about data. The data that is used to represent other data is known as metadata. For example, the index of a book serves as a metadata for the contents in the book. In other words, we can say that metadata is the summarized data that leads us to detailed data. In te

Tutorialspoint

Backlog

Topics

Fault Tolerant Systems:

- General Reliability Development Hazard logs (FRACAS) [Redundancia]
- High Availability [https://www.controlglobal.com/assets/14WPpdf/140324-ISA-ControlSystemsHighAvailability.pdf]
- Technical documentation
- Safety cases
- Bulkhead
- Change Control
- Cold Standby
- Defensive Design
- Derating
- Design Debt
- Design Life
- Design Thinking
- Durability
- Edge Case
- Entropy
- Error Tolerance
- Fault Tolerance
- Fail Well
- Fail-Safe
- Graceful Degradation
- Mistake Proofing & Poka Yoke Technique
- No Fault Found
- Resilience
- Safety by Design
- Self-Healing

- Service Life
- Systems Thinking
- Testbed
- Waer and Tear
- Deconstructability
- Refinement
- Defense in Depth
- FMEA Design and Process
- Physics of Failure (PoF)
- Built-in Self-test
- Eliminating single point of failure (SPOF)

Analysis:

- Root Cause analysis
- Fault tree analysis (FTA)
- Failure mode and effects analysis (FMEA)
- Failure mode, effects and criticality analysis (FMECA)
- Reliability, Availability and Maintainability Study (RAMS)
- Mission Readiness analysis
- Functional System Failure analysis
- Inherent Design Reliability analysis
- Use/Load analysis and wear calculations
- Fatigue and creep analysis
- Component Stress analysis
- Field failure monitoring
- Field data analysis
- Caution and warning analysis
- Chaos Engineering
- Reliability Risk Assessments
- Hazard analysis
- Manufactoring defect analysis
- Residual Risk analysis (RCA)
- Weibull
- Accelerated Life Testing (ALT Analysis)
- Material Strength analysis
- Quality of Service

- Quality Control
- Defect Rate
- Failure Rate
- Mean Time Between Failures
- Mean Time to Repair (MTTR)
- Mean Corrective Maintenance Time (MCMT)
- Mean Preventive Maintenance Time (MPMT)
- Mean Maintenance Hours per Repair (MMH/Repair)
- Maximum Corrective Maintenance Time (MaxCMT)

Data Quality:

- Data Quality Completeness
- Data Quality Correctness
- Data Quality Credibility
- Data Quality Precision
- Data Quality Relevance
- Data Quality Timeliness
- Data Quality Traceability
- Data Integrity
- Data Cleansing
- Data Corruption
- Data Degradation
- Data Artifact
- Data Rot
- Information Quality Accurate
- Information Quality Completeness
- Information Quality Comprehensible
- Information Quality Credibility
- Information Quality Precision
- Information Quality Relevance
- Information Quality Timeliness
- Information Quality Uniqueness
- Conformance Quality
- Credence Quality
- Quality Assurance
- Quality Control

- Service Quality
- Experience Quality
- Code Smell
- Referential Integrity
- Reusability

Maintenance:

- Maintenance Requirement Allocation
- Predictive and Preventive maintenance
- Reliability Centered Maintenance (RCM)

Failures:

- Manufactoring-induced failures
- Assembly-induced failures
- Transport-induced failures
- Storage-induced failures
- Systmatic failures

Tests:

- System Diagnostics Design
- Failure/Reliability testing

Human Factors:

- Human Factors
- Human Interaction
- Human Errors
- Latent Human Error

DataOps:

Business Process Management:

- BPM
- BPI
- BPE
- BPA

- BPR