Según Sommerville la **Ingeniería de Software** se define como aquella que se refiere a los problemas prácticos de producir software. Y, según Pressman, es la aplicación de enfoques sistémicos, disciplinados y cuantificables al desarrollo, operación y mantenimiento de software y el estudio de estos enfoques.

La **metodología** para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito.

El **proceso de software** es un conjunto estructurado de actividades necesarias para desarrollar un sistema de software. Es una abstracción de un proceso real y presenta una descripción desde una perspectiva particular.

Según la ISO, define al **ciclo de vida** de un software como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando desde la definición hasta la finalización de su uso.

Descripción de cada una de las **etapas** del ciclo de vida del software:

- Análisis: se determinan los elementos que intervienen en el sistema a desarrollar, su estructura, relaciones, evolución temporal, funcionalidades.
- Diseño: se define cómo se llevará a cabo, se entra en detalle entidades y relaciones de las bases de datos, se selecciona el lenguaje, entre otros.
- Implementación: se codifican algoritmos y estructuras de datos en el lenguaje de programación o para un determinado gestor de bases de datos.
- Debugging/Prueba: garantiza que no haya errores de diseño o codificación.
- Mantenimiento: es el agregado de funcionalidad y corrección de errores que hayan surgido.

Además:

- Validación: indica que el sistema desarrollado cumple con los requerimientos del cliente.
- Verificación: indica si los requisitos están bien recogidos y si los productos de cada fase cumplen con los impuestos sobre ellos en las fases previas.

Tipos de ciclo de vida:

- ciclo de vida lineal: es el más sencillo, descompone la actividad global del proyecto en etapas separadas realizadas de manera lineal (las etapas se realizan una sola vez). Sus ventajas es que es fácil de dividir tareas y prever tiempos, sencillez de gestión, administración, economía y temporal. En cuanto a desventajas, no tiene retroalimentación lo que conlleva excesiva rigidez en saber lo que hará cada etapa y es muy costoso retomar un paso anterior cuando se detectan fallas. Ejemplo: los programas pequeños de ABM (alta baja modificación), una empresa que desea registrar datos para luego consultarlos.

- ciclo de vida en cascada: admite iteraciones, cada fase genera documentación para la siguiente. De ventajas tiene planificación sencilla y cierta flexibilidad para regresar a una etapa anterior. En cuanto a desventajas es un modelo rígido, no es tan flexible y tiene muchas restricciones, los errores se detectan en etapas finales siendo costoso y retrasa el tiempo, es difícil de mantener, es poco realista. Ejemplos: a lo mejor se puede aplicar cuando el ciclo lineal no sea tan adecuado y utilizar un modelo más elaborado no lo justifique.
- ciclo de vida en V: contiene las mismas etapas que el anterior, pero se le agregaron dos subetapas de retroalimentación (Validación entre Análisis y Mantenimiento, Verificación entre Diseño y Debugging). Posee las mismas ventajas y desventajas del anterior, solo se agrega en ventajas los controles cruzados entre etapas para mayor corrección. Ejemplos: pequeñas transacciones sobre bases de datos, una aplicación de facturación.
- ciclo de vida tipo Sashimi: se pueden solapar las etapas lo cual aumenta la eficiencia por la retroalimentación entre etapas. Ventajas es la ganancia de calidad, no es necesario una documentación detallada. Como desventaja también se nombra el solapamiento ya que es difícil gestionar el comienzo/fin de cada etapa, y si hay problemas de comunicación se generan inconsistencias. Ejemplos: aplicaciones que comparten recursos como una CPU, memoria o espacio de almacenamiento.
- ciclo de vida en espiral: se basa en repetir ciclos hasta que el cliente/usuario obtiene la satisfacción de sus necesidades. Entre sus ventajas es que puede comenzarse el proyecto con un alto grado de incertidumbre, bajo riesgo de retraso en caso de errores, es flexible. Desventajas es el costo temporal, la dificultad para evaluar los riesgos y la necesidad de comunicación continua con el cliente. Ejemplos: aplicación que administre reclamos, pedidos e incidentes.

La **Ingeniería de Requerimientos** es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos del sistema.

Los **requerimientos** son una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. Los requerimientos según Sommerville, es una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. (lo dejo para el que lo quiera estudiar).

Características de los requerimientos:

- necesario: lo es si su omisión provoca una deficiencia en el sistema a construir, y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o proceso.
- conciso: lo es si es fácil de leer y entender, de redacción simple y clara.
- completo: lo está si se proporciona la información suficiente para su comprensión.

- consistente: lo es si no es contradictorio con otro requerimiento.
- no ambiguo: no lo es cuando tiene una sola interpretación. El lenguaje usado no debe causar confusiones.
- verificable: lo es cuando puede ser cuantificado y permite hacer uso de inspección, análisis, demostración o pruebas.

Dificultades de los requerimientos: no son obvios y vienen de muchas fuentes, son difíciles de expresar en palabras, existen muchos tipos y de diferente nivel de detalle, la cantidad de estos en un proyecto lo hace difícil de manejar, nunca son iguales, están relacionados unos con otros, tienen propiedades únicas y abarcan áreas específicas, pueden cambiar a lo largo del proyecto.

Tipos de requerimientos:

- requisitos funcionales: definen las funciones que el sistema será capaz de realizar, describen las transformaciones, a lo largo del proyecto éstos se convierten en los algoritmos, la lógica y gran parte del código.
- requisitos no funcionales: tienen que ver con las propiedades que puedan limitar al sistema, como el rendimiento, interfaz de usuario, fiabilidad, seguridad, etc.

Técnicas de recolección de información:

- entrevistas: son útiles para despejar dudas específicas y presupone cierto conocimiento previo del tema, requieren planificación.
- JAD (Joint Application Design): es un conjunto de reuniones entre usuarios y analistas, se comienza con un documento de trabajo y se obtiene un documento de requisitos aprobados.
- prototipado: consiste en presentarle al cliente una maqueta para su experimentación con algunas de las características del sistema final
- observación: se conoce la forma en que se realizan las tareas, salen fácilmente las condiciones alternativas en el curso, el objetivo final es perceptible, las necesidades reales del usuario son fáciles de comprender.
- estudio de documentación: revisar manuales, especificaciones o registros existentes.
- cuestionarios: son formularios estructurados para recopilar datos de varias personas rápidamente.
- Brainstorm (tormenta de ideas): generación de ideas creativas en grupo, no censura ideas y las relaciona entre sí,
- casos de uso: secuencia de interacciones entre un sistema o solución y el usuario.

El **Análisis de Requisitos** es un proceso de estudio y refinamiento de los mismos. Sus actividades generales son:

- extracción: se utilizan las técnicas de recolección de información.
- análisis: razonamiento sobre ellos, descubrimiento de inconsistencias, etc.
- especificación: registro de los mismos, en lenguaje natural o usando técnicas.
- validación: el cliente comprueba que son correctos.

El **Estudio de Factibilidad** es un análisis que realiza una empresa para saber si el proyecto se va a poder desarrollar. Sus objetivos son reducir el tiempo de desarrollo, mejorar servicios a los clientes, reducir costos mediante la eliminación de recursos innecesarios y saber si es posible producir ganancias. En este participan el analista de sistemas, el personal de auditoría y RRHH (área contable).

Tipos de factibilidad:

- económica: trata de determinar la conveniencia de poner en marcha el proyecto, incluye el análisis de costos y beneficios asociados a cada alternativa del proyecto.
- técnica: trata de demostrar la posibilidad de concreción del proyecto, evalúa si el equipo y software están disponibles y cumplen con las capacidades requeridas para cada alternativa.
- operativa: busca verificar que el sistema será capaz de operar con éxito, comprende las características que un sistema utilice según lo esperado.

Si los tres tipos de factibilidad se cumplen, el proyecto es viable por lo que puede llevarse a cabo.

Los **árboles de decisión** son diagramas que pretenden mostrar la gama de posibles resultados y las decisiones posteriores realizadas después de la decisión inicial. Lleva a cabo una evaluación a medida que se recorre hacia las hojas para alcanzar así una decisión, y suele contener nodos internos, de probabilidad, hojas y arcos. Éstos ayudan a las empresas a determinar cuáles son sus opciones al mostrarles las distintas decisiones y sus resultados.

Sus ventajas son:

- obligación de tener tantos resultados posibles de una decisión como te puedas imaginar.
- plantean el problema para que todas las opciones sean analizadas.
- ayuda a realizar las mejores decisiones sobre la base de la información existente y suposiciones.

Sus desventajas son:

- los resultados, decisiones y pagos se basarán fundamentalmente en las expectativas.
- los eventos inesperados pueden alterar las decisiones y cambiar los pagos.
- requieren un gran número de datos de los que muchas veces no disponemos.

Los tipos de árboles de decisión son: de clasificación, de regresión, de mejora, bosques de árboles de decisión, de clasificación y regresión, agrupamiento de las K medias.

Una **tabla de decisión** es una herramienta que sirve para representar de manera más fácil la lógica de un problema cuando es más o menos complicada. Para ello se trata de identificar en el problema las acciones que hay que ejecutar y las condiciones que se tienen que cumplir para ejecutar esas acciones. Sus partes son el conjunto de condiciones, entrada de condiciones, conjunto de acciones, salida de ejecución y la regla de decisión.

Se construye definiendo las condiciones, determinando las acciones posibles, determinando las alternativas para cada condición, calcular el máximo de columnas (2^n), armar la tabla de cuatro cuadrantes, determinar las reglas y completar alternativas, completarla y eliminar redundancia.

Tipos de tablas de decisión:

- de entrada limitada/binaria: las condiciones se expresan en forma de preguntas porque está limitada a dos valores, por ejemplo V-F, si-no.
- de entrada extendida: las condiciones pueden tomar más de dos valores.
- de entrada mixta: se combinan las anteriores, considerando los valores de las condiciones de entrada extendida e identificando las acciones de la limitada o viceversa.

Las tablas de decisión son herramientas que se utilizan:

- en la etapa de análisis para efectar una representación gráfica simplificada de los procesos lógicos, a fin de analizar si se adecuan o no a los requerimientos.
- en la etapa de diseño para respresentar gráficamente procesos lógicos creados para satisfacer las necesidades del sistema.
- aisladamente, es decir, para representar simplificadamente procedimientos específicos que sirvan de apoyo para la interpretación del sistema.

La **utilidad** permite representar la descripción de situaciones decisivas, es decir, se representan las distintas alternativas, estados de la naturaleza y las consecuencias. También proporcionan una descripción completa, correcta, clara y concisa de una situación que se resuelve por una decisión tomada en un tiempo determinado.

Los **diagramas de flujo de datos** son utilizados para modelar la funcionalidad de un sistema, proporcionan una representación del sistema a nivel lógico y conceptual. Permite representar un sistema como una red de procesos de transformación de datos que intercambian información por medio de flujos de datos (definición más formal).

Sus elementos básicos son la entidad externa que representa entes ajenos a la aplicación, son los que aportan o reciben información; el proceso que es la actividad

que transforma o manipula datos; el almacén de datos que es el depósito de información dentro del sistema; los flujos de datos que establecen la comunicación entre procesos, almacenes y entidades externas, llevan la información necesaria.

Se basan en el principio de descomposición o explosión por niveles, comúnmente se utiliza para análisis top-down (de general a particular). Un proceso que no puede descomponser se le llama Proceso Primitivo. El nivel 0 es un diagrama de contexto, el nivel 1 son los subsistemas y el nivel 2 son funciones de cada subsistema; además el nivel 3 son subfunciones asociadas y el nivel 4 son procesos necesarios para el tratamiento de cada subfunción.

Objetivos: la declaración se destina al cliente y otras personas que no están involucradas directamente en el desarrollo del sistema, no está destinado a dar una descripción detallada y abarcativa, establece el ámbito del sistema sin margen de duda y con la menor cantidad posible de palabras, formulado de manera abstracta y conceptual, comienzan con un verbo.
Límites: establece qué elemetos están dentro y fuera del sistema.

Alcances: estudia el alcance de la necesidad planteada por el cliente, se analizan las posibles restricciones, tanto globales como específicas, que puedan condicionar el estudio e interferir en la planificación.

La **entidad** es un objeto o concepto que tiene una existencia independiente.
Los **atributos** son propiedades específicas que describen la entidad. Pueden ser simples o compuestos (divididos en subpartes). Pueden ser monovaluado o multivaluado que tiene un conjunto de valores.
La **relacion** es la asociación entre varias entidades, pueden tener atributos descriptivos.
La **cardinalidad** es el número de entidades con las que se puede asociar otra entidad a través de relaciones. Pueden ser (1,1), (1,N), (N,1) y (N,N).
La **superclave** es un conjunto de uno o más atributos que permiten identificar de forma única a una entidad.
Las **claves candidatas** son superclaves para las cuales ningún subconjunto propio es superclave.
La **clave primaria** es aquella clave candidata que elige el diseñador de la base de datos.

El **modelo entidad - relación** se usa para describir datos a nivel conceptual, está basado en objetos, está compuesto por entidades, atributos y relaciones, con sus etiquetas y cardinalidades correspondientes.

El **modelo relacional** es la representación lógica del esquema entidad - relación. Se basa en el concepto de tablas compuestas de requisitos y campos.

Una **base de datos** es un conjunto de datos que pertenecen al mismo contexto, almacenados sistemáticamente para su posterior uso. Debe cumplir las propiedades ACID que garantizan la fiabilidades e integridad de las transacciones.
A por Atomicidad que asegura que la transacción se realice en su totalidad o no se realice en absoluto. C por Consistencia que asegura que la transacción lleve la BD de un estado válido a otro estado válido, manteniendo reglas y restricciones. I por Aislamiento que asegura que las transacciones concurrentes se ejecuten independientes y sin interferencias entre sí. D por Durabilidad que asegura que una vez finalizada la transacción sus efectos persisten en la BD.

La **normalización** de BD es un principio de diseño para organizar los datos de una manera consistente y estructurada, su propósito es evitar complejidades, eliminar datos duplicados y organizar los datos. La primer forma normal establece que la tabla debe tener una clave primaria, la segunda forma normal establece que los atributos no clave deben depender de la clave completa y la tercera forma normal establece que ninguna columna no clave debe depender transitivamente de la clave primaria.

El concepto de **ocultamiento de información** sugiere que los módulos se caracterizan por decisiones de diseño que se ocultan de las demás. Deben especificarse y diseñarse módulos de forma que la información construida en un módulo sea inaccesible para las que no necesiten de ella.

La **independencia (de módulos) funcional** es el resultado directo de la separación de problemas y de los conceptos de abstracción y ocultamiento de información.

Desarrollo de software (rol del analista de sistemas):

- recolección de registros: documentar las necesidades del cliente.
- analista de sistemas: determina cómo se estructuran los componentes del sistema.
- especificaciones funcionales: crear un documento detallado de cómo debe funcionar el sistema.
- validación de requerimientos: asegurarse de que las especificaciones sean viables técnica y económicamente.

Documentación es el proceso mediante el cual los desarrolladores convierten los registros y especificaciones funcionales en códigos que la computadora pueda comprender.

1) escribir el código: implementar las funcionalidades del sistema en uno o más lenguajes de programación.
2) depuración y pruebas: verificar que el código funciona como se espera y, en caso contrario, solucionar errores.
3) optimización: mejorar el rendimiento y la eficiencia del software.

Las **categorías del lenguaje de programación** son lenguaje de alto nivel (python, java, php) que son fáciles de leer y escribir ya que están más cerca al lenguaje humano y lenguaje de bajo nivel (ensamblador) que están más cerca del lenguaje máquina ofreciendo mayor control sobre el software.

Las **pautas de análisis y programación** son:

- modularidad: consiste en dividir el sistema en componentes más pequeños y manejables.
- reutilización de código: implementar componentes genéricas que pueden ser reutilizables.
- documentación: asegurar que todo el código y las decisiones de diseño están debidamente documentadas.
- pruebas exhaustivas: realizar pruebas unitarias, de integración y de sistema.

Pruebas de software es el proceso de ejecutar un programa con el objetivo de encontrar errores. Las pruebas no sólo buscan validar que el sistema cumpla con los requisitos especificados, sino también descubrir fallas o comportamientos inesperados. Según Sommerville hay puntos a tener en cuenta:

- objetivos: el principal objetivo de las pruebas no es demostrar que el sistema funcione correctamente, sino que encontrar errores.
- validación y verificación: la validación va a permitir mostrar que cumple con los requisitos del cliente y verificar que sea implementado correctamente según el diseño.
- calidad de software: es una actividad clave para garantizar que el software cumple con los requisitos, identificando defectos antes de que lleguen al cliente.

Tipos de prueba:

- Caja negra: se centra en las entradas y salidas del software sin considerar cómo funcionan internamente. El tester no tiene acceso al código fuente y prueba el sistema basándose en los requisitos funcionales.
- Caja blanca: se enfoca en el conocimiento interno del código fuente y la estructura del programa. Se basa en la lógica del código y permite al tester verificar cómo funcionan los componentes internos del mismo.
- Pruebas Unitarias: son aquellas que validan que cada función/método individual del código funcione bien de manera aislada.
- Pruebas de Integración: se encarga de verificar que diferentes módulos/componentes del sistema funcionen bien cuando se integren entre sí.
- Pruebas de Carga: es un tipo de prueba de rendimiento que verifica cómo se comporta el sistema bajo una carga normal o alta esperada.
- Pruebas de Stress: se enfoca en verificar cómo se coporta el sistema cuando se somete a una carga extrema o por encima de sus capacidades máximas.

ejemplo de prueba de carga: una web espera recibir 15 mil clientes simultáneos durante el black friday, la prueba simula el número de clientes para ver cómo la web maneja la demanda y si se mantiene estable dentro de los parámatros esperados.
ejemplo de prueba de stress: con el ej anterior, podría simular 30 mil clientes simultáneos, que es el doble esperado, para ver en qué punto el sistema comienza a fallar.

¿cuándo se dice que una prueba es exitosa? cuando se encontró al menos un error.
intervienen en las pruebas el desarrollador, usuario, gerente y auditor.
Prueba de Volumen: consiste en analizar el funcionamiento del sistema en grandes volúmenes de datos.
Prueba de Regresión: verifica que los cambios en el código no afecten las funcionalidades existentes.

Clasificación de las pruebas:

- funcionales: el objetivo es validar que el sistema realice las funciones especificadas en los requerimientos.
- no funcionales: el objetivo es validar aspectos que tengan que ver con calidad del sistema, rendimiento, seguridad, usabilidad, etc.

El **error** ocurre por una falla humana al diseñar y proponer el sistema.
El **defecto** es el resultado del error en el sistema. Es un comportamiento no esperado.
La **falla** es una caída del sistema que le impide completar su misión en tiempo de operación.
ejemplo de error: un desarrollador olvida programar una excepción "para dividir por 0" en la calculadora.
ejemplo de defecto: en una aplicación de calculadora el código que maneja las divisiones no considera casos en los que el denominador es 0.
ejemplo de falla: un usuario intenta dividir un número por 0 y la aplicación se cierra inesperadamente o devuelve un resultado incorrecto.

Axiomas "se hace en cualquier programa":

1) el software siempre se testea.
2) un buen caso de testeo es el que muestra que el software no anda, no el que funciona correctamente sin completar su misión.
3) nada es más difícil que dejar de testear.
4) a medida que avanza el testeo, crece la posibilidad de no encontrar errores.
5) el mejor usuario debe realizar el testeo de caja negra.
6) el mejor desarrollar debe realizar el testeo de caja blanca.