REQUISITO: es una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. En la ingeniería clásica, los requisitos se utilizan como datos de entrada en la etapa de diseño del producto, establecen QUÉ DEBE HACER el sistema, pero no CÓMO HACERLO.

¿Qué es un requisito?

- condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo. → IEEE ← condición o capacidad que un sistema necesita para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta.
- condición o capacidad que debe ser conformada por el sistema → RUP
- algo que el sistema debe hacer o una cualidad que el sistema debe poseer → Robertson-Robertson

En ingeniería de software existen 3 tipos de requisitos:

- requisito funcional: descripción de lo que un sistema o sus componentes deben hacer, establecen el comportamiento del software. Una funcion es un conjunto de entradas, comportamientos y salidas. Ej: calculos, detalles técnicos, manipulación de datos. Contiene: nombre, número de serie único y un resumen.
- requisito no funcional o atributo de calidad: son los criterios que pueden usarse para juzgar la operación de un sistema en lugar de su comportamiento, establecen las características de funcionamiento. Ej: tiempo de entrega, rendimiento, calidad, lenguaje, programa, cantidad de usuarios.
- limitaciones externas: afectan de forma indirecta al producto. Ej: compatibilidad del SO, regulaciones y leyes.

Pseudorrequisitos: aquellos referidos al entorno donde será instalado o implementado el sistema, que determinan en gran medida su desarrollo, pueden ser cuestiones como HW y SW.

Características (suelen ser subjetivas, no pueden ser calculadas de forma automática por ningún sistema. Se utilizan métricas o indicadores para ponderar las características)

- No ambiguo: el texto deber ser claro, preciso y tener una única interpretación posible,
- Conciso: redactarse en un lenguaje comprensible por los inversores en lugar de uno de tipo técnico y especializado, aunque aún así debe referenciar los aspectos importantes.
- Consistente: ningún requisito debe entrar en conflicto con otro requisito diferente, ni con parte de otro. Asimismo, el lenguaje entre los distintos requisitos debe ser consistente.
- Completo: Los requisitos deben contener en sí mismos toda la información necesaria, y no remitir a otras fuentes externas que los expliquen con más detalle.
- Alcanzable: Un requisito debe ser un objetivo realista, posible de ser alcanzado con el dinero, el tiempo y los recursos disponibles.
- Verificable: Se debe poder verificar con absoluta certeza, si el requisito fue satisfecho o no. Esta verificación puede lograrse mediante inspección, análisis, demostración o testeo.

Metodología: determina los pasos a seguir y cómo realizarlos para finalizar una tarea, con el fin de mejorar la productividad en el desarrollo y la calidad del producto software.

Etapas (CICLO DE VIDA): la ingeniería del software requiere llevar a cabo numerosas tareas agrupadas en etapas.

1. Obtención de requisitos: identificar qué motiva el inicio del estudio y creación del nuevo software o modificación de uno ya existente. A su vez identificar los recursos humanos y materiales que participan en el desarrollo de las actividades. Para ello es importante entender el contexto del negocio (las necesidades definidas por la alta dirección, tener dominio del problema, conocer datos fuera del software como usuario final, otros sistemas, también datos que salen del sistema por la interfaz de usuario u otros medios, y conocer los almacenamiento de datos) y la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos de software. Reconocer los puntos críticos, aquellos que entorpecen y limitan el buen funcionamiento de los procedimientos actuales y los motivos. Para ello define el comportamiento ante situaciones inesperadas.

0. Previo, puede haber una fase de análisis conceptual que incluye la recolección de requisitos, análisis de consistencia e integridad, definición en términos descriptivos para los desarrolladores y un esbozo de especificación.

2. La etapa donde se estudian los requisitos para verificar que estén correctamente adecuados a las características mencionadas es la de análisis de requisitos, en la misma se intentan solucionar las deficiencias que los requisitos pueden tener. En esta etapa el cliente plantea la necesidad e intenta explicar lo que deberia hacer el software, mientras que el desarrollador actúa como interrogador. Muchos clientes piensan que saben todo lo que el software necesita para su buen funcionamiento, sin embargo se requiere la habilidad y experiencia de algún especialista para reconocer requisitos incompletos, ambiguos o contradictorios. El resultado del análisis de requisitos se plasma en el documento ERS (especificación de requisitos del sistema), cuya estructura puede venir definida por varios estándares, como CMMI, asimismo se define un diagrama de entidad/relación en el que se plasma las principales entidades. La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales; para ello se idearon modelos y diversos procesos metódicos de trabajo, aunque no está formalizado, se habla de ingeniería de requisitos.

Finalidades del análisis de requisitos:

- brindar al usuario todo lo necesario para que pueda trabajar en conjunto con el software desarrollado obteniendo los mejores resultados posibles.
- tener un control más completo en la etapa creación de software, en cuanto a tiempo de desarrollo y costos (modelo COCOMO).
- utilización de métodos más eficientes que permitan el mejor aprovechamiento del software según su finalidad de uso.
- aumentar la calidad del software desarrollado al disminuir los riesgos de mal funcionamiento.

Limitaciones: el software posee funciones predefinidas que abarcan un conjunto de soluciones que en algunos campos llega a ser limitado. Además, proviene del proceso

totalmente mecánico que requiere un mayor esfuerzo y tiempos elevados de ejecución, lo que lleva a tener que implementar el software en una máquina de mayor capacidad.

Técnicas de especifiación de requisitos:
- caso de uso
- historias de usuario

3. Arquitectura: el arquitecto de software es quien añade valor a los procesos de negocios con soluciones tecnológicas. Es una actividad de planeación, ya sea a nivel de infraestructura de red y hw o de sw. Lo principal es poner en claro los aspectos lógicos y físicos de las salidas, modelos de organización y representación de datos, entradas y procesos que componen el sistema, considerando las limitaciones de los recursos disponibles. Elabora un plan de trabajo viendo la prioridad de tiempo y recursos disponibles, y se encuentra la interpretación de requerimientos (dominio de la información del problema, funciones para el usuario, comportamiento del sistema).
El diseño arquitectonico permite visualizar la interacción entre las entidades del negocio (componentes de la aplicación), para ello se utilizan diagramas: de clase, de base de datos, de despliegue, de secuencia. Las herramientas para el diseño y modelado de software se denominan CASE (computer aided software engineering).

4. Programación: no necesariamente es la que demanda más trabajo, depende del lenguaje de programación y el diseño previamente realizado.

5. Desarrollo de la aplicación: es necesario considerar 5 fases para tener un programa eficiente.
- desarrollo de la infraestructura: permite el desarrollo y la organización de los elementos que formarán la infraestructura.
- adaptación del paquete: entender el funcionamiento del paquete garantiza que pueda ser utilizado en su máximo rendimientos, tanto para negocios o recursos.
- desarrollo de unidades de diseño interactivas: se realizan los procedimientos que se ejecutan por un diálogo usuario-sistema, tienen como objetivo establecer las acciones que debe efectuar la unidad de diseño, la creación de componentes para sus procedimientos y la ejecución de pruebas unitarias y de integración.
- desarrollo de unidades de diseño batch: se utilizan diagramas de flujo, de estructuras, tablas de decisiones, etc. para plasmar las especificaciones de manera clara.
- desarrollo de unidades de diseño manuales: el objetivo es proyectar todos los procedimientos administrativos en torno a la utilización de los componentes.

6. Pruebas de software: una técnica es probar cada módulo del software (pruebas unitarias) y luego probar de manera integral (pruebas de integración). Es una buena práctica que las pruebas sean efectuadas por alguien distinto al desarrollador. Hay 2 maneras, que esté compuesto por personal que desconoce el tema de pruebas, para asegurar que la documentación es clara para cualquiera y el software hace las cosas tal cual, el segundo es tener un área de pruebas conformada por programadores con experiencia, por lo que pueden poner atención a detalles. Es importante que durante el proceso de desarrollo no se pierda contacto con los interesados para tener una idea clara de los aspectos que tienen que probarse.

7. Implementación: es el proceso de convertir una especificación del sistema en un sistema ejecutable. Muchas especificaciones son dadas según un estándar (WWWC world wide web consortium). El modelo de implementación es una colección de componentes (ficheros) y los subsistemas que contienen.

8. Documentación: UML, diagrama de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc. todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.

9. Mantenimiento: puede llevar más tiempo que el desarrollo del software inicial, alrededor de ⅔ del ciclo de vida de un proyecto. Consiste en eliminar errores y principalmente (80%) extender el sistema.

Ventajas
Desde el punto de vista de gestión
- Facilitar la tarea de seguimiento del proyecto
- Optimizar el uso de recursos
- Facilitar la comunicación entre usuarios y desarrolladores
- Facilitar la evaluación de resultados y cumplimiento de objetivos

Desde el punto de vista de los ingenieros de software
- Ayudar a comprender el problema
- Permitir la reutilización
- Facilitar el mantenimiento del producto final
- Optimizar el conjunto y cada una de las fases del proceso de desarrollo

Desde el punto de vista de cliente o usuario final
- Garantizar el nivel de calidad del producto final
- Obtener el ciclo de vida adecuado para el proyecto
- Confianza en los plazos del tiempo mostrados en la definición del proyecto

CASO DE USO: concepto creado en 1986 por Jacobson, mejoras por Cockburn en el 2000. entidades=actores (operadores humanos, sistemas externos, entidades abstractas). Representa un conjunto de interacciones. Se usan para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. Es un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo. Es común para la captura de requisitos funcionales. Tipos de relaciones:
- comunica: denota participacion del actor en caso de uso
- usa o incluye: dependencia entre dos casos de uso → se repite el comportamiento
- generalización: un caso de uso es una variante de otro, variando cualquier aspecto de uso base.
- extiende: el caso de uso base declara puntos de extensión → se presenta una variación del comportamiento
- también pueden existir relaciones de herencia entre casos de uso o actores.

Modelos de caso de uso: combinación de casos de uso y sus correspondientes diagramas. Suelen estar acompañados de un glosario que describe la terminología utilizada.

Normas de aplicación: evitan el lenguaje técnico, prefieirndo el lenguaje del usuario final. Son elaborados en colaboración por los analistas de requisitos y los clientes. Cada caso de uso se centra en una única meta o tarea, un caso de uso describe una caracteristica del sistema. Contiene una descripcion textual de todas las maneras que los actores podrían trabajar con el software o el sistema. No describen funcionalidades internas ni implementación.
En la fase de extracción el analista se concentra en las tareas centrales del usuario describiendo por lo tanto los casos de uso que mayor valor aportan al negocio, esto facilitará la priorización de requisitos.
Aunque se asocia a la fase de test, es erróneo, su uso se extiende mayormente a las primeras fases de un desarrollo.

Limitaciones: útiles para establecer requisitos de comportamiento, pero no establecen completamente los requisitos funcionales ni permiten determinar los requisitos no funcionales. Cada caso crítico de uso debe tener un requisito no funcional centrado en el funcionamiento asociado.

HISTORIAS DE USUARIO: describen el por qué y qué detrás del desarrollo además del valor que provee al usuario final, es una explicación general informal de un componente del software descrita desde la perspectiva del usuario final. Los clientes o usuarios finales no tienen que ser externos de la organización. Son oraciones que explican el resultado deseado sin ir a detalle, los requerimientos son añadidos después. Se usan en metodologías ágiles como Scrum o Kanban, ayudan a obtener mejores estimaciones y manejar WIP (work in progress).
Se utilizan porque mantienen el foco en el usuario real, permiten la colaboración al estar el objetivo final definido, se puede decidir entre todos cuál es la mejor opción, además las historias de usuario permiten soluciones creativas, ya que obliga al equipo a pensar de manera crítica y creativa.
Las historias de usuario las suele hacer el product owner, product manager o program manager. Durante los sprints se discuten los requerimientos y funcionalidades, y se puntúa basado en complejidad y time of completion (se usa Fibonacci).

¿Cómo escribir historias de usuario? definition of done, outline subtasks or tasks, user personas (para quién, si hay multiples usuarios finales, considerar hacer multiples historias), ordered steps, listen to feedback (de los usuarios), time (como las historias deben ser completadas en un sprint, si son muy largas, conviene dividirlas en más chicas). Estructura: “As a [persona], I [want to], [so that].”