Requisito: necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. Se utilizan como datos de entrada en la etapa de diseño del producto, establecen qué debe hacer el sistema pero no cómo

* Condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo
* Condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta
* Una condición o capacidad que debe ser conformada por el sistema
* Algo que el sistema debe hacer o una cualidad que el sistema debe poseer

Hay tres tipos de requisitos:

* Requisito funcional: establecen los comportamientos del sw. Especifica una función (conjunto de entradas, comportamientos y salidas) que el sistema entregado debe ser capaz de realizar. Pueden ser cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas. Los requisitos de comportamiento se muestran en los casos de uso. Se generan luego de realizar los casos de uso
  + Tiene nombre, número de serie y resumen
* Requisito no funcional (atributos de calidad): especifica algo sobre el propio sistema y cómo debe realizar sus funciones. No describen información a guardar ni funciones a realizar, sino características de funcionamiento. Ejemplos: tiempo de entrega, lenguaje, cantidad de usuarios
* Otras limitaciones externas que afectan de forma indirecta al producto. Puede ir desde ser compatibles con algún sistema operativo hasta la adecuación a leyes o regulaciones aplicables al producto

En la etapa de análisis de requisitos se intenta solucionar las deficiencias que tienen.

Una colección de requisitos describe las características o atributos del sistema deseado. No se define cómo se hace la implementación porque eso es en la etapa de diseño.

Pseudorrequisitos: referidos al entorno donde será instalado o implementado el sistema. Determinan en gran medida su desarrollo (podría ser hw o sw).

Características: no pueden ser calculadas automáticamente por sistemas, por lo que se tiende a usar métricas e indicadores que sí pueden ser calculadas por sistemas

* No ambiguo: texto claro, preciso y solamente una interpretación posible
* Conciso: redactados en lenguaje comprensible por los inversores en vez de uno técnico y especializado. Debe referenciar los aspectos importantes
* Consistente: ningún requisito debe entrar en conflicto con otro diferente ni con parte de otro. El lenguaje usado entre los requisitos debe ser consistente
* Completo: deben contener en sí toda la información necesaria. No deben remitir a fuentes externas que los expliquen con más detalle
* Alcanzable: debe ser un objetivo realista, posible de ser alcanzado con el dinero, tiempo y recursos disponibles
* Verificable: se debe poder verificar con absoluta certeza si fue satisfecho o no. Se puede verificar con inspección, análisis, demostración o testeo

Metodología: pasos sistemáticos, predecibles y repetibles que mejoran la productividad y la calidad del sw. Tiene diferentes etapas, el conjunto de ellas se llama ciclo de vida

* Obtención de requisitos: identificar sobre qué se está trabajando, motivo por el cual se estudia y crea (o modifica) el sw, así como el contexto del negocio. También se identifican los recursos (humanos y materiales) que se tiene. Se debe saber la información del problema, incluido los datos fuera del sw (usuarios finales, sistemas, dispositivos externos); los que salen del sistema (interfaz de usuario, interfaces de red, reportes, gráficas); y los almacenamientos de datos que guardan y organizan objetos persistentes de datos. Se deben ver los puntos críticos (aspectos que entorpecen y limitan el buen funcionamiento de los procedimientos, problemas comunes y relevantes, motivos de insatisfacción, puntos que deben ser cubiertos a plenitud). Se deben definir las funciones del sw tanto para el usuario final como para su funcionamiento. Tener en cuenta el comportamiento del sw ante situaciones inesperadas
* Análisis de requisitos: el cliente a veces piensa que sabe todo lo que necesita el sw, pero se necesita de un analista experimentado para saber requisitos incompletos, ambiguos o contradictorios. Los requisitos son en base a las necesidades del usuario final, usando técnicas para mejorar la calidad de los sistemas. El resultado del análisis es el documento ERS (especificación de requisitos del sistema) y se define un diagrama de entidad/relación para tener las principales entidades del sistema. Las finalidades del análisis de requisitos son:
  + Brindar al usuario todo lo necesario para que pueda trabajar en conjunto con el sw desarrollado obteniendo los mejores resultados posibles
  + Tener un control más completo en la etapa de creación del sw, en cuanto a tiempo y costo
  + Utilización de métodos más eficientes que permitan el mejor aprovechamiento del sw según sea la finalidad de uso
  + Aumentar la calidad del sw al disminuir los riesgos de mal funcionamiento
* Limitaciones: los sw emulan inteligencia a partir de ciertas características de la inteligencia humana, pero solo tiene un conjunto de soluciones limitados. Actúa bajo condiciones. Otra limitación es el proceso mecánico que requiere mayor esfuerzo y tiempos elevados de ejecución, lo que lleva a implementar el sw en una máquina de mayor capacidad
* Especificación: describe el comportamiento esperado del sw. Se debe identificar las necesidades del negocio y la interacción con los usuarios para la recolección, clasificación, identificación, priorización y especificación de los requisitos
* Arquitectura: integración de infraestructura, desarrollo de aplicaciones, bases de datos y herramientas gerenciales; teniendo en cuenta los aspectos lógicos y físicos de las salidas, modelos de organización y representación de datos, entradas y procesos que componen el sistema. El arquitecto aporta valor a los procesos a través de soluciones tecnológicas, es una actividad de planeación. En base a la arquitectura del sistema se elabora un plan de trabajo viendo la prioridad del tiempo y recursos disponibles; en los diseños de salida está la información del problema, funciones visibles para el usuario, comportamiento del sistema y un conjunto de clases que agrupa los objetos del negocio con los métodos que les dan servicio
  + La arquitectura de sw es el diseño de componentes de una app usando patrones de arquitectura. Debe permitir visualizar la interacción entre entidades y se debe poder validar. Se documenta utilizando diagramas (ejemplos: diagrama de clases, diagrama de bases de datos, etc)
* Programación: implementar el diseño en código. La complejidad y duración está relacionada a los lenguajes de programación
* Desarrollo de la app: 5 fases
  + Desarrollo de la infraestructura: desarrollo y organización de la infraestructura
  + Adaptación del paquete: entender de forma detallada el funcionamiento del paquete, para que el mismo pueda ser usado en su máximo rendimiento (para negocios o recursos). Se inspeccionan detalladamente los elementos que forman el paquete para entenderlos y ver que no haya errores
  + Desarrollo de unidades de diseño interactivas: se realizan los procedimientos que se ejecutan en un diálogo usuario-sistema, los objetivos son:
    - Establecer específicamente las acciones que debe efectuar la unidad de diseño
    - Creación de componentes para sus procedimientos
    - Ejecutar pruebas unitarias y de integración en la unidad de diseño
  + Desarrollo de unidades de diseño batch: combinación de técnicas (como diagrama de flujo, diagrama de estructuras, tablas de decisiones, etc) para definir clara y objetivamente las especificaciones, para que el programador entienda mejor a la hora de programar y testear
  + Desarrollo de unidades de diseño manuales: proyectar todos los procedimientos administrativos que se desarrollarán en torno a la utilización de los componentes computarizados
* Pruebas de sw: comprobar que el sw realiza correctamente las tareas especificadas en el problema. Una forma es hacer pruebas unitarias para cada módulo y luego pruebas de integración (probar todo junto). Es buena práctica que las haga alguien diferente al desarrollador, en un área de pruebas, para no verse influenciado por lo que sabe del código. Para el área de pruebas, se usa un grupo inexperto de personal para verificar que la documentación es clara; el segundo es un grupo experimentado de programadores que sabe en qué condiciones podría fallar el sistema. Las pruebas sirven para los procesos lógicos internos del sw y los procesos externos funcionales (detectar errores). Mantener el contacto con los interesados para tener claro qué cosas probar
* Implementación: realización de una especificación técnica o algoritmos con un programa, componente u otro sistema de cómputo. Esta etapa es el proceso de convertir una especificación en un sistema ejecutable. Implica el diseño y programación de sw, podría implicar un refinamiento. Es una descripción de la estructura del sw que se va a implementar, los datos que son parte del sistema, interfaces entre componentes y a veces los algoritmos usados
* Documentación: documentar el desarrollo del sw y la gestión del proyecto, usando modelos, diagramas, casos de usos, pruebas, manuales de usuario, manuales técnicos, etc. Sirve para correcciones, usabilidad, mantenimiento y ampliaciones
* Mantenimiento: mantener y mejorar el sw para corregir errores descubiertos e incorporar requisitos nuevos, puede llevar más tiempo que el desarrollo inicial. Alrededor de 2/3 del tiempo del ciclo de vida está dedicado al mantenimiento. El 80% son mejoras de funcionalidad (el resto es corregir bugs), dado que el sw evoluciona con el tiempo. Se reutiliza el sw existente, reformándolo y adaptándolo

Casos de uso: descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. Los personajes o entidades que participan en un diagrama son actores. Se representa un sistema como un conjunto de interacciones entre casos de uso y actores en respuesta a un evento que inicia un actor principal. Los diagramas sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con usuarios y otros sistemas (relación entre actores y casos de uso), también la reacción a eventos que se producen en sí mismo o en el ámbito. Normalmente se usan para ilustrar los requisitos funcionales.

* Actores: entidad externa al sistema que guarda una relación con este y que le demanda una funcionalidad (operadores humanos, sistemas externos, entidades abstractas como el tiempo). En el caso de humanos, se definen como rol, por lo que un individuo puede corresponder a varios actores. En el caso del tiempo, es un requisito funcional ejecutar operaciones automáticas en determinados momentos
* Relación: conexión entre elementos del modelo.
  + Tipos:
    - Comunica (<<communicates>>): relación entre un actor y un caso de uso que denota la participación del actor en ese caso
    - Usa (<<uses>>, <<include>>): relación de dependencia entre dos casos que denota la inclusión del comportamiento de un escenario en otro
    - Generalización: un caso es una variante de otro. El caso especializado puede variar cualquier aspecto del caso base
    - Extiende (<<extend>>, <<extiende>>): dependencia, forma de extensión más controlada que la relación de generalización. El caso base declara un conjunto de puntos de extensión, el caso especializado solo puede alterar el comportamiento de los puntos de extensión marcados
  + <<extends>> se usa cuando hay casos similares pero uno hace más que el otro. <<uses>> cuando hay comportamientos similares pero no se quiere repetir la descripción de dicho comportamiento común
  + En una relación <<extends>>, el actor que use al caso base puede realizar o no extensiones. En <<include>>, el actor que realiza el caso de uso base también realiza el caso de uso incluido
  + En general <<extends>> cuando hay una variación del comportamiento normal. <<include>> cuando se repite un comportamiento en dos casos y se quiere evitar la repetición
  + Pueden existir relaciones de herencia entre casos o entre actores
  + Modelo de casos de uso: combinación de casos y sus diagramas. Los modelos de casos de uso tienen un glosario que describe la terminología. El glosario y el modelo sirven para el desarrollo de diagramas de clase
  + Aunque cada caso puede llevar a diferentes realizaciones, se debe reflejar el motivo que llevó a descartarla si se da
* Pasos para la definición de un caso de uso:
  + ID
  + NOMBRE
  + REFERENCIAS CRUZADAS
  + CREADO POR
  + ÚLTIMA ACTUALIZACIÓN POR
  + FECHA DE CREACIÓN
  + FECHA DE ÚLTIMA ACTUALIZACIÓN
  + ACTORES
  + DESCRIPCIÓN
  + TRIGGER
  + PRE-CONDICIÓN
  + POST-CONDICIÓN
  + FLUJO NORMAL
  + FLUJOS ALTERNATIVOS
  + INCLUDES
  + FRECUENCIA DE USO
  + REGLAS DE NEGOCIO
  + REQUISITOS ESPECIALES
  + NOTAS Y ASUNTO
* Normas de aplicación:
  + Escritos en lenguaje natural o el lenguaje preferido del cliente, no en lenguaje técnico. A menudo son elaborados por los analistas de requisitos y los clientes
  + Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea
  + Describe una característica del sistema. Para definir un sistema, quizás es necesario definir muchos casos de uso
  + El grado de formalidad y la etapa del proyecto puede influir en el nivel de detalle requerido para el caso de uso
  + Contiene una descripción textual de las maneras en las que los actores podrían trabajar con el sistema
  + No describen funcionalidades internas (ocultas al exterior) del sistema ni cómo se implementarán. Muestra lo que debe hacer el actor para usar la funcionalidad
  + Debe ser bastante sencillo como para que un desarrollador lo elabore en un único lanzamiento
  + Un actor puede comunicarse con un caso de uso
  + Un caso de uso puede extender a otro
  + Un caso de uso puede utilizar a otro
* Facilidades:
  + Sirve en sistemas interactivos porque muestran la intención del actor
  + Permite al analista centrarse en las necesidades del cliente para la funcionalidad en vez de dejar que los desarrolladores lo hagan en base a criterios tecnológicos
  + Permite priorizar según lo que aporte más valor al negocio, facilitando las prioridades de los requisitos
* Limitaciones:
  + No establecen completamente los requisitos funcionales ni requisitos no funcionales
  + Se deben complementar con información adicional como reglas de negocio, requisitos no funcionales, diccionarios de datos que complementen los requisitos del sistema, etc
  + Cada caso crítico de uso debe tener un requisito no funcional centrado en el funcionamiento asociado

Historias de usuario: explicación general e informal de una funcionalidad desde el punto de vista del usuario final, sirven para ver cómo le aporta valor al usuario.

* Mínima unidad de trabajo en entornos ágiles de trabajo
* Ayudan a enfocarse en el usuario final, trabajando en equipo para ver las mejores soluciones para alcanzar esa meta
* Generalmente escrita por el producto owner, producto manager o program manager, se manda a revisión
* En el desarrollo, se eligen qué historias hacer en base a prioridades
* Para escribirlas:
  + Definition of “done”: definir cuándo el usuario puede decir que está hecho
  + Definir pasos previos para terminar con la historia, asignarlos a personas
  + User personas: para quién es la historia, si son muchos hacer varias
  + Para los procesos largos, hacer historias para cada paso que se toma
  + Escuchar feedback
  + Historias grandes que van a llevar más de un sprint, separarlas en historias más chicas
  + Ejemplo: as a [persona], I [want to\*], [so that].
    - Persona: para quién se está construyendo
    - Want to: la intención, no la funcionalidad usada. No debería haber nada de implementación acá
    - So that: objetivo general de hacer eso, por qué quieren hacer esa acción
    - As Max, I want to organize my work, so I can feel more in control
    - As a manager, I want to be able to understand my colleagues progress, so I can better report our success and failures