Resúmen de Lectura Básica UT1

Requisitos

https://es.wikipedia.org/wiki/Requisito_(sistemas)

Requisito es una necesidad documentada sobre un producto o servicio. Establece qué debe hacer el sistema y no cómo hacerlo.

Condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo. Algo que el sistema debe hacer o capacidad que debe poseer.

Tres tipos de requisitos en Ingeniería en Sistemas:

- Requisito funcional: descripción de lo que el sistema debe hacer. Algo que el sistema debe ser capaz de realizar. "el sistema debe permitir a los usuarios iniciar sesión en su cuenta" o "el sistema debe calcular la factura de un cliente y aplicar el impuesto correspondiente"
  Se relacionan directamente con el software y se puede probar en su 100% si se cumple.

- Requisito no funcional: de rendimiento, de calidad, especifica algo sobre el propio sistema y cómo debe realizar sus funciones. Ejemplos: disponibilidad, testeo, mantenimiento, facilidad de uso. “El sistema debe ser capaz de manejar 1000 usuarios simultáneamente" o "el sistema debe estar disponible para el uso del usuario durante el horario de oficina".
  Se relacionan con la calidad del software y no se pueden probar directamente.

- Otros tipos de limitaciones externas, que afectan de una forma indirecta al producto. Ejemplo: compatibilidad con sistema operativo, cambio de leyes.

Una colección de requisitos describe las características del sistema deseado. Se omite el cómo puede lograrse su implementación.

Características:

- No ambiguo: claro y preciso.
- Conciso: lenguaje comprensible, no técnico.
- Consistente: no puede entrar en conflicto con otro requisito.
- Completo: deben tener toda la información necesaria.
- Alcanzable: debe ser realista.
- Verificable: se debe verificar al 100% si fue satisfecho o no.

La etapa de requisitos es Análisis de requisitos.

Metodologías

https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software#Metodolog%C3%ADa

La ingeniería de software requiere llevar a cabo numerosas tareas agrupadas en etapas. Al conjunto de etapas se denomina ciclo de vida. Las etapas comunes a casi todos los ciclos de vida son las siguientes:

Obtención de los requisitos: se identifica sobre que se está trabajando (tema principal), identificar los recursos que se tienen. Se entiende el contexto del negocio para poder identificar los requisitos. Comprender las necesidades y expectativas de los usuarios finales y los interesados en el software.

Análisis de Requisitos: en este paso se analiza la información recolectada del paso anterior, se definen los requisitos del sistema en un conjunto de requisitos funcionales y no funcionales, para asegurarse de que el software cumpla con las necesidades.
A veces los clientes creen saber todo el funcionamiento que necesita el software, pero, requiere la habilidad de un especialista para reconocer requisitos incompletos.

Limitaciones: el software tiene la capacidad de emular inteligencia, pero solo posee modelos creados que abarcan un conjunto de soluciones, por lo que puede llegar a ser limitado. No es capaz de emular el pensamiento humano.

Especificación: la especificación de requisitos es esencial para el éxito del software ya que describe el comportamiento del software una vez desarrollado. La identificación de las necesidades de negocio es fundamental para recolectar, clasificar, identificar, priorizar y especificar los requisitos del software.
Técnicas de especificación de requisitos: **Casos de uso y las historias de usuario.**

Arquitectura: la Arquitectura de Software se enfoca en el diseño de componentes de una aplicación usando patrones de arquitectura y herramientas CASE (Computer Aided Software Engineering). El diseño arquitectónico se documenta utilizando diagramas: diagramas de clase, bases de datos, despliegue y secuencia. Y debe permitir la visualización y validación entre las entidades del negocio.
En esta etapa se definen los aspectos estructurales y organizacionales del software, incluyendo la identificación de componentes del sistema, su interconexión y las interfaces entre ellos.

Programación: la duración está relacionada al diseño previamente realizado.

Desarrollo de la aplicación: 5 fases para tener una aplicación eficiente:

- Desarrollo de la infraestructura (desarrollo de elementos que formarán la infraestructura)
- Adaptación del paquete (se examina en detalle el paquete (conjunto de software), sus ventajas y desventajas y se decide si usarlo, modificarlo o cambiarlo).
- Desarrollo de unidades de diseño interactivas (se realizan todas las unidades de diseño que impliquen funcionalidad y se hacen pruebas unitarias y de integración)

- Desarrollo de unidades de diseño batch (se utilizan una combinación de técnicas: diagrama de flujo, diagramas de estructuras, tablas de decisiones, de manera de plasmar de manera precisa las especificaciones así el equipo de programadores comparte la misma idea)
- Desarrollo de unidades de diseño manuales (se enfoca en la creación de los procesos administrativos que se desarrollarán en torno a la utilización de componentes computarizados de la aplicación).

Pruebas de software: el objetivo de las pruebas es detectar y corregir errores y problemas antes de que el software sea puesto en producción.

Implementación: es el proceso de convertir las especificaciones en un sistema ejecutable.

Documentación: documentación del propio software, modelación UML, diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos.

Mantenimiento: arreglar errores (bugs) y extender el sistema para agregar funcionalidades. 80% son mejoras de funcionalidad mientras que el otro 20% son solucionar errores.

Caso de uso

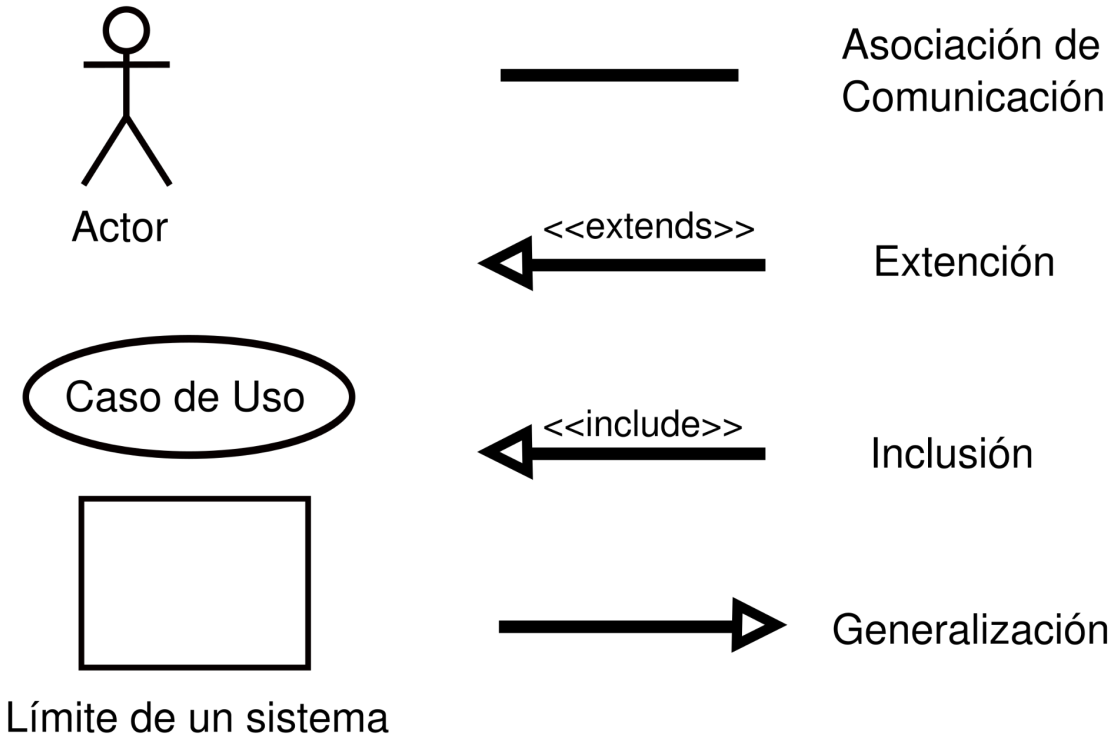
https://es.wikipedia.org/wiki/Caso_de_uso

Es una técnica utilizada en ing. Software para describir cómo un usuario interactúa con la aplicación, es una descripción detallada de una acción o tarea de un usuario en un sistema y cómo el sistema responde a esa acción.
En un caso de uso se describe quién es el usuario que realiza la acción, qué acción realiza y qué respuesta espera del sistema. Por lo general, se presenta en forma de un diagrama de casos de uso que muestra los diferentes usuarios del sistema, las acciones que pueden realizar y las respuestas esperadas.

Los casos de uso son una herramienta valiosa para la definición de los requisitos del sistema y la identificación de escenarios de uso. Se usa principalmente para POO pero puede ser para otro paradigma.

Actores => personajes que aparecen en el caso de uso. Entidad externa al sistema que le demanda una funcionalidad (humanos, el tiempo, sistemas externos, cosas abstractas).

Relación => conexión entre los elementos del modelo, especialización y generalización son relaciones**.**



Tipos de relaciones:

- Comunica (<<comunicates>>): relación (asociación) entre un **actor** y un **caso de uso**, y representa la participación de un actor en un caso de uso.
- Usa (<<uses>> o <<include>>): es una relación de dependencia **entre dos casos de uso**. Indica que un caso de uso utiliza el comportamiento definido de otro caso de uso. Por ejemplo, el caso de uso “realizar compra” puede incluir/usar el caso de uso “procesar pago”.
- Generalización: indica que un **caso de uso es una variante de otro**. El caso de uso especializado puede variar cualquier aspecto del caso de uso base. Por ejemplo “comprar boleto de avión” el caso especializado sería “comprar boleto de avión con ventanilla”, pero mantienen un parentesco.
- Extiende (<<extend>>): es un estereotipo de **dependencia que ofrece una forma más controlada** de extender un caso de uso base. Extend permite que el caso de uso especializado altere unos puntos de extensión marcados en el caso de uso base.

Los casos de uso son una herramienta de ingeniería de software que se utiliza para describir cómo un sistema debe funcionar desde la perspectiva del usuario final o del experto del campo de aplicación. Son elaborados en colaboración por analistas de requisitos y clientes y se centran en describir una única meta o tarea. Un caso de uso describe una característica del sistema y puede ser necesario especificar decenas o centenares de casos de uso para definir completamente el nuevo sistema. El grado de formalidad del proyecto y la etapa del proyecto influirán en el nivel de detalle requerido en cada caso de uso.

Los casos de uso no describen ninguna funcionalidad interna oculta del sistema, sino que muestran lo que el actor hace o debe hacer para realizar una operación. Un caso de uso debe describir una tarea del negocio que sirva a una meta de negocio, tener un nivel apropiado de detalle y ser lo suficientemente sencillo como para que un desarrollador lo elabore en un único lanzamiento.

Las situaciones que pueden darse incluyen que un actor se comunique con un caso de uso (si se trata de un actor primario la comunicación la iniciará el actor, en cambio si es secundario, el sistema será el que inicie la comunicación), un caso de uso extienda otro caso de uso y un caso de uso utilice otro caso de uso.

Requisito funcional

https://es.wikipedia.org/wiki/Requisito_funcional

Un **requisito funcional** define una función del sistema de software o sus componentes.
Una función es descrita como un conjunto de entradas, comportamientos y salidas.
Los requisitos funcionales pueden ser:

- Cálculos
- Detalles técnicos
- Manipulación de datos
- Otras funcionalidades específicas que un sistema debe cumplir

Son los que establecen los comportamientos del software. Y son complementados por los requisitos no funcionales.
Los requisitos de comportamiento para cada requisito funcional se muestran en los casos de uso. Generalmente se realizan primero los casos de uso y luego los requisitos funcionales, pero pueden existir excepciones debido a que el desarrollo de software es un proceso iterativo y algunos requisitos son previos al diseño de casos de uso. Ambos elementos se complementan en un proceso bidireccional.
Un **requisito funcional** típico contiene un **nombre**, un **número de serie** único y un **resumen**. Lo cual ayuda al lector a entender porque el requisito es necesario y para seguir al mismo durante el desarrollo del producto.
El **núcleo** de los requisitos yace en la descripción del comportamiento requerido, la cual debe ser clara y concisa. Este comportamiento puede prevenir de reglas organizacionales o del negocio, o ser descubiertas por interacción con usuarios, inversores y otros expertos en la organización.

Requisito no funcional

https://es.wikipedia.org/wiki/Requisito_no_funcional

Un **requisito no funcional** o también conocido como **atributo de calidad**, es un requisito que especifica **criterios** que pueden usarse **para juzgar la operación de un sistema** en lugar de sus comportamientos específicos (requisitos funcionales).
Se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino **características de funcionamiento**.

Son las **restricciones o condiciones** que impone el cliente al programa que necesita, por ejemplo, el tiempo de entrega del programa, el lenguaje o la cantidad de usuarios.

Categorías:

- Requisitos de calidad de ejecución, incluye seguridad, usabilidad y otros medibles en tiempo de ejecución.
- Requisitos de calidad de evolución, como testabilidad, extensibilidad o escalabilidad, los cuales se evalúan en los elementos estáticos del sistema de software.

Ejemplos:

- Rendimiento
- Disponibilidad
- Durabilidad
- Estabilidad
- Accesibilidad
- Adaptabilidad
- Documentación
- Mantenibilidad
- Seguridad
- Eficiencia
- Reusabilidad
- Compatibilidad

Historias de usuario

https://www.atlassian.com/agile/project-management/user-stories

Las historias de usuario son **tareas de desarrollo** usualmente expresadas como “persona + necesidad + propósito”

Una historia de usuario es una **explicación general e informal** de una función de software escrita desde la **perspectiva** del **usuario final o cliente**. Su propósito es articular **como una función** de software le **proveerá valor al cliente**. El cliente no siempre es un usuario final externo, sino que también puede ser un cliente interno o un colega dentro de la organización el cual depende de tu equipo.

Una historia de usuario **no es lo mismo que un requisito**.

Un **componente clave** del desarrollo ágil de software es poner a las **personas primero**, las historias de usuario **centran el enfoque** en el **usuario final**. Estas historias utilizan un **lenguaje no técnico** para proporcionar un **contexto** al equipo de desarrollo. Luego de leer una historia de usuario el equipo conoce **porque** están construyendo, **que** están construyendo y **que valor** brinda.

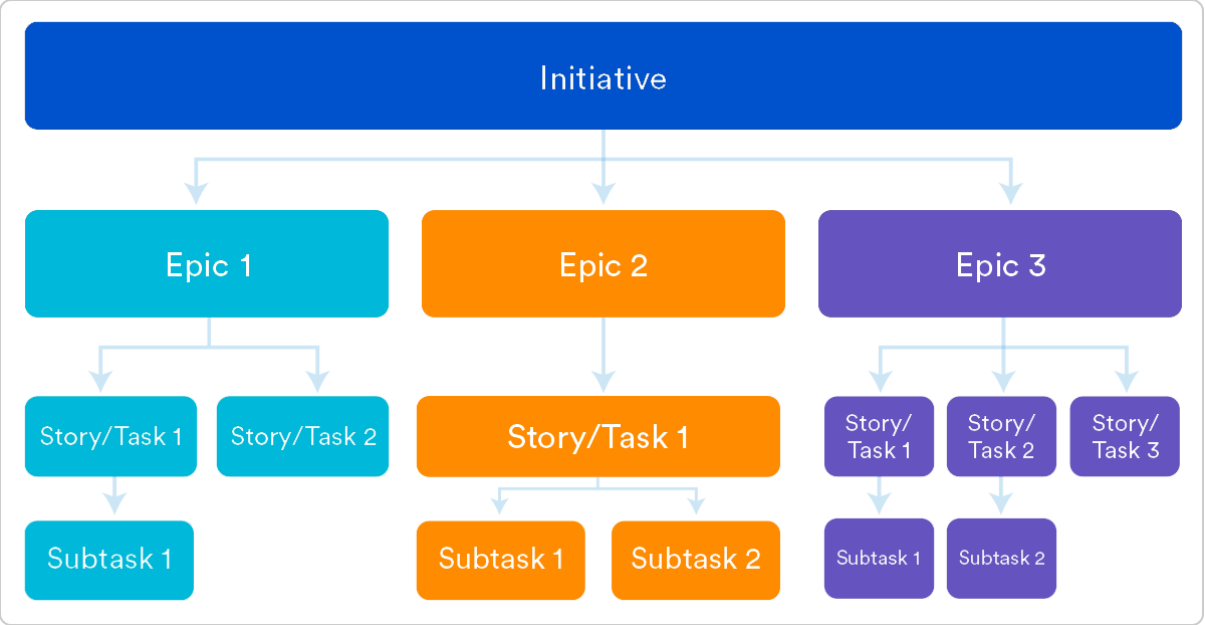
Las historias de usuario son uno de los **componentes centrales** para un programa ágil. Estas ayudan a proveer un framework de uso diario centrado en el usuario, lo que impulsa la colaboración, la creatividad y un mejor producto.

Historias de usuario ágiles
Una historia de usuario es la unidad de trabajo más pequeña en un framework ágil.
Es un objetivo final, no una función, expresada desde la perspectiva del usuario.

Las historias de usuario son unas **pocas oraciones** en lenguaje simple que resaltan el **resultado esperado**. No entran en detalles, los requisitos son agregados luego.

Las historias de usuario encajan perfectamente con frameworks ágiles como Scrum y Kanban.

Las historias de usuario son a su vez los componentes básicos de frameworks ágiles más grandes como **epics** y **initiatives**. **Epics** son grandes ítems de trabajo desglosados en un conjunto de historias, y múltiples epics constituyen una **initiative**.
Estas estructuras más grandes aseguran que el trabajo diario del equipo de desarrollo contribuya a los objetivos organizacionales integrados en las epics y initiatives.



Por que crear historias de usuarios?
Las historias de usuario brindan al equipo de desarrollo un contexto importante y asocia las tareas con el valor que brindarán.

Las historias de usuario brindan una serie de beneficios claves:

- **Mantienen el enfoque en el usuario**, una lista de tareas mantiene al equipo enfocado en tareas a realizar, mientras que, una colección de historias mantiene al equipo enfocado en resolver problemas para usuarios reales.
- **Facilitan la colaboración**, con el objetivo final definido, el equipo puede trabajar en conjunto para definir la mejor manera de servir al usuario y a su vez lograr el objetivo.
- **Conducen a soluciones creativas**, alientan al equipo a pensar de manera crítica y creativa sobre cómo resolver un objetivo final de la mejor manera posible.

- **Crean momentum (impulso)**, con cada historia que pasa, el equipo de desarrollo disfruta de un pequeño desafío y una pequeña victoria, creando impulso.

Trabajando con historias de usuario

Una vez la historia de usuario ha sido escrita, es tiempo de integrarla al flujo de trabajo. Generalmente una historia es escrita por el dueño del producto, el manager del producto o el manager del programa y es enviada para ser revisada.

Durante una reunión de planificación de *sprint* o *iteración*, el equipo decidirá que historias se trabajarán en ese *sprint*. Donde se discutirán los requisitos y las funcionalidades que cada historia de usuario requiere.

La decisión de qué historias se trabajarán en cada *sprint* se toma teniendo en cuenta una estimación de la **carga horaria y de trabajo** que cada historia presenta. Una historia debería ser **completada en un solo *sprint***, por lo que, en el caso de existir historias de usuario muy extensas, se debe dividir la misma en distintas historias, las cuales se puedan abarcar en un solo *sprint*.

Como escribir historias de usuario

- **Definición de "terminada" –** Una historia está "terminada" cuando el usuario puede realizar la tarea especificada, por lo que se debe definir claramente cual es.
- **Resaltar tareas y subtareas –** Decidir cuáles pasos específicos se deben completar y quien es responsable de completarlos.
- **Personas de usuario –** ¿Para quién está pensada la historia? Si existen múltiples usuarios finales se considera el realizar múltiples historias.
- **Pasos ordenados –** Escribir una historia para cada paso en un proceso mayor.
- **Escuchar el feedback –** Hablar con los usuarios y capturar el problema o necesidad en sus palabras. No hay necesidad de adivinar al crear historias cuando se puede obtener la información de los clientes.
- **Tiempo –** El tiempo es un tema delicado. Por lo que muchos equipos de desarrollo suelen evitarlo. Dado que las historias deben ser completadas en un *sprint*, historias que llevan semanas o meses deben ser divididas en diferentes historias o ser consideradas como *epics*.

Una vez que las historias de usuario estén definidas, se deben hacer visibles a todo el equipo.

Template y ejemplos de historias de usuario

Usualmente las historias de usuario son expresadas en una simple oración.
"Como [una persona], yo [quiero algo], [de manera que]."

- "Como [una persona]": ¿Para quien estamos construyendo esto? No solo buscamos un título de trabajo, buscamos la personalidad de la persona. Max. Nuestro equipo debe tener una comprensión compartida de quién es Max. Debemos entender como

esa persona trabaja, como piensa y lo que siente. Debemos sentir empatía con esa persona.

- “Quiero algo”: Aquí describimos sus intenciones, no las funcionalidades que usa. ¿Qué es lo que está tratando de lograr? No debemos pensar en implementaciones, si en esta parte se describe cualquier aspecto de la interfaz gráfica y no cual el objetivo del usuario realmente es, estamos equivocados.
- “De manera que”: ¿Cómo encaja su deseo inmediato de hacer algo así en el panorama general? ¿Cuál es el beneficio general que están tratando de lograr? ¿Cuál es el gran problema que necesita solución?

Algunos ejemplos:

- Como Max, quiero invitar a mis amigos, de manera que podamos disfrutar del servicio juntos.
- Como Pedro, quiero organizar mi trabajo, para poder sentirme en control.
- Como Lucía, quiero ser capaz de entender el progreso de mis colegas, para poder reportar de mejor manera nuestros éxitos y fracasos.

Esta estructura no es obligatoria, pero es de gran ayuda para poder definir correctamente cuando una historia de usuario está ”terminada”.