Ingeniería de requisitos

Breve descripción:

En este componente formativo se abordan los saberes de ingeniería de requisitos: ciclo de vida del “software” fases y objetivos, modelos, características, caracterización de la fase de definición de requisitos y herramientas para el uso de captura de requisitos que se usan para el desarrollo del “software”.

Marzo 2024

Tabla de contenido

Introducción

Bienvenidos al estudio del presente componente formativo, en el cual se estudiará la ingeniería de requisitos, entendiéndola como el conjunto de actividades y tareas del proceso de desarrollo de sistemas “software”, donde se definen que las características de un sistema “software” satisfagan las necesidades de negocio de clientes y usuarios y que se integre con éxito en el entorno en el que se trabaje.

Igualmente, gestiona las líneas base y las peticiones de cambios que se vayan produciendo en la especificación de requisitos, manteniendo la trazabilidad entre los requisitos y otros productos del desarrollo.

Por este motivo, estudiaremos el ciclo de vida del “software”, sus fases y los diferentes paradigmas de los modelos de vida del “software”, haciendo énfasis en los requisitos, su importancia y clasificación.

Muchos éxitos en esta experiencia de aprendizaje.

1. Ciclo de vida del “software”

También conocido como (SDLC o “Systems Development Life Cycle”), es el proceso que se sigue para construir y hacer evolucionar un determinado “software”. El ciclo de vida permite iniciar una serie de fases mediante las cuales se procede a la validación y al desarrollo del “software” garantizando que se cumplan los requisitos para la aplicación y verificación de los procedimientos de desarrollo; para ello, se utilizan métodos del ciclo del “software”, que indican distintos pasos a seguir para el desarrollo de un producto.

Si bien existen diferentes ciclos de desarrollo de “software”, la normativa ISO/IEC/IEEE 12207:2017 establece que:

Es un marco común para los procesos del ciclo de vida de los programas informáticos, con una terminología bien definida, a la que pueda remitirse la industria del “software”. Contiene procesos, actividades y tareas aplicables durante la adquisición, el suministro, el desarrollo, el funcionamiento, el mantenimiento o la eliminación de sistemas, productos y servicios informáticos. Estos procesos del ciclo de vida se llevan a cabo mediante la participación de los interesados, con el objetivo final de lograr la satisfacción del cliente (s.p.).

A continuación, se indican cuáles son los elementos que integran un ciclo de vida:

- **Fases.** Una fase es un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se construye agrupando tareas (actividades elementales) que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales

correspondientes a la asignación de recursos (humanos, financieros o materiales).

- **Entregables**. Son los productos intermedios que generan las fases. Pueden ser materiales o inmateriales (documentos, "software"). Los entregables permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales y de condiciones de realización previamente establecidos.

1.1. Fases

Las fases del modelo de ciclo del "software" son: planificación, análisis, diseño, implementación, pruebas y mantenimiento, las cuales se describen a continuación.

- **Fase Planificación:**

  En esta primera fase se realiza el planteamiento del problema, se definen alcances y objetivos del "software".

  Objetivos: estudio de viabilidad, realizar planificación detallada.

- **Fase Análisis (definición de requisitos):**

  Esta fase busca definir los requisitos que son los que dirigirán el desarrollo del proyecto de "software".

  Objetivos: conocer los requisitos, asegurar que los requisitos son alcanzables, formalizar acuerdo con el cliente.

- **Fase Diseño:**

  En esta fase se estudian posibles opciones de implementación para el "software" que hay que construir, estructura general del mismo.

Objetivos: identificar soluciones tecnológicas, asignar recursos materiales, proponer identificar y seleccionar, establecer métodos de validación, ajustar especificaciones.

- **Fase Pruebas:**

  Esta fase busca detectar fallos cometidos en las etapas anteriores para corregirlos.

  Objetivos: realizar los ajustes necesarios para corregir posibles errores o inconsistencias.

- **Fase Mantenimiento:**

  En esta fase se realizan tres puntos referenciados: mantenimiento correctivo, mantenimiento adaptativo y mantenimiento perfectivo.

  Objetivos: operación, asegurar que el uso del proyecto es el que se pretendía, mantenimiento.

1.2. Paradigmas de los modelos de ciclo de vida del "software"

Con la finalidad de proporcionar una metodología común entre el cliente y la empresa de "software", se utilizan los modelos de ciclos de vida o paradigmas de desarrollo de "software" para plasmar las etapas y la documentación necesaria, de manera que cada fase se valide antes de continuar con la siguiente.

Un modelo de ciclo de vida de "software" es una vista de las actividades que ocurren durante el desarrollo de "software" e intenta determinar el orden de las etapas involucradas y los criterios de transición asociados entre estas.

Según la norma 1074 IEEE se define al ciclo de vida del "software" como "una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del "software"".

La ISO/IEC 12207 "Information Technology" / "Software Life Cycle Processes" señala que es:

Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de "software", abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso (2008).

Existen modelos preestablecidos con los cuales se puede elaborar un proyecto; a continuación, se mencionan los diferentes paradigmas de modelos de ciclo de vida para desarrollar "software".

- **Paradigma tradicional**

  Los paradigmas tradicionales se identifican, fundamentalmente, por ser lineales, es decir, se trata de completar cada proceso de principio a fin, hasta que quede listo para avanzar a la segunda fase del ciclo del "software".

  Si bien es verdad que las metodologías actuales están basadas en lo que fueron los paradigmas tradicionales, hoy en día se ha evolucionado, sin embargo, los paradigmas tradicionales se mantienen.

  Desventaja: Pérdida de tiempo si se encuentran errores en una fase avanzada, porque al devolverse, se debe pasar nuevamente por todas las fases y reestructurar, de acuerdo con las modificaciones.

- **Paradigma orientado a objetos**

  Las etapas de desarrollo de "software" en el paradigma orientado a objetos, se conforma principalmente por la creación de clases, análisis de requisitos y el diseño. Con este paradigma se pretende que el código fuente sea reutilizable para otros proyectos.

- **Paradigma de desarrollo ágil**

  El objetivo de este paradigma es el desarrollo de proyectos en poco tiempo, se simplifican procesos tediosos, se agilizan las fases del desarrollo y las interacciones se hacen en corto tiempo.

  Una de las principales diferencias con los paradigmas anteriores, es que el cliente se ve involucrado en el proyecto durante el desarrollo de este, así, el cliente sugiere mejoras, propone ideas y se mantiene al tanto del desarrollo del producto, a diferencia del paradigma tradicional y el orientado objeto, donde el cliente únicamente está al principio.

A continuación, se revisan los modelos del paradigma tradicional más utilizados.

- **Modelo en cascada**

  Uno de los primeros modelos de ciclo de vida del desarrollo fue establecido por W. Royce en 1970 y es conocido como el "modelo de cascada" ("waterfall model").

  En su concepción básica, cada una de las actividades genera, como salidas, productos y modelos que son utilizados como entradas para el proceso subsiguiente; esto supone que una actividad debe terminarse (por lo menos, en algún grado) para empezar la siguiente.

- **Modelo espiral**

Fue diseñado por Boehm en el año 1988 y se basa en una serie de ciclos repetitivos para ir ganando madurez en el producto final. El espiral se repite las veces que sea necesario hasta que el cliente o el usuario obtenga la satisfacción de sus necesidades.

En este modelo hay 4 actividades que envuelven a las etapas: planificación, análisis de riesgo, implementación y evaluación. Una de sus principales ventajas es que los riesgos van disminuyendo conforme avanzan los ciclos o interacciones.

- **Modelo iterativo o por prototipos**

  Este modelo consiste en un procedimiento que permite al equipo de desarrollo diseñar y analizar una aplicación que represente el sistema que será implementado (McCracken y Jackson, 1982).

  Objetivos

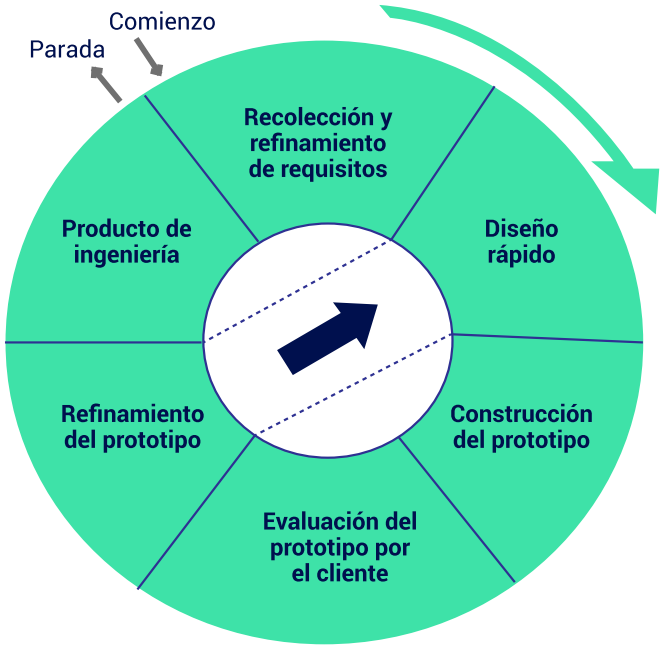
  - Son un medio eficaz para aclarar los requisitos de los usuarios e identificar las características de un sistema que deben cambiarse o añadirse.
  - Mediante el prototipo se puede verificar la viabilidad del diseño de un sistema.

  Las etapas del modelo son:

  - Colecta y refinamiento de los requerimientos y proyecto rápido.
    - Análisis.
    - Especificación del prototipo.
  - Diseño rápido.
  - Construcción del prototipo.

- Evaluación del prototipo por el cliente.
- Refinamiento del prototipo.
  - Diseño técnico.
  - Programación y “test”.
  - Operación y mantenimiento.
- Producto de ingeniería.

Figura 1. Modelo iterativo o por prototipos



Con respecto a los modelos del ciclo de vida del paradigma ágil, estos se caracterizan por estar basados en etapas del ciclo de vida del “software” tradicional, pero combinándolas con algunas técnicas, al respecto se pueden revisar los siguientes:

Modelo “Scrum”

Este modelo se basa en el desarrollo incremental, es decir conforme pasen las fases y la iteración mayor será el tamaño del proyecto que se está desarrollando.

Los procesos que utiliza son:

- “Product Backlog”.
- “Sprint Backlog”.
- “Sprint Planning Meeting”.
- “Daily Scrum”.
- “Sprint Review”.
- “Sprint Retrospective”.

El “Scrum” consiste en realizar un análisis de los requerimientos del sistema (“Product Backlog”), señalar cuáles serán los objetivos a corto o mediano plazo dentro de un sprint, o sea, la fase de desarrollo. Posteriormente, los desarrolladores harán lo suyo, se realizarán algunas pruebas y se retroalimentará de acuerdo con lo conseguido al terminar la última fase.

Ventajas

- **Gestión regular de las expectativas del usuario**: los usuarios participan y proponen soluciones.
- **Resultados anticipados:** no es necesario esperar hasta el final para ver resultados.
- **Flexibilidad y adaptación:** se adapta a cualquier contexto, área o sector.
- **Gestión sistemática de riesgos:** los problemas son gestionados en el mismo momento de su aparición.

Modelo Kanban

David J. Anderson (reconocido como el líder de pensamiento de la adopción del Lean/Kanban para el trabajo de conocimiento), formuló el método Kanban como una aproximación al proceso evolutivo e incremental y al cambio de sistemas para las organizaciones de trabajo. El método está enfocado en llevar a cabo las tareas pendientes y los principios más importantes pueden ser divididos en cuatro principios básicos y seis prácticas.

El modelo Kanban es uno de los modelos más visuales de las metodologías ágiles; este consiste en la creación de un tablero con etiquetas, donde se seccionan cada una de las fases de su desarrollo, además se clasifican de acuerdo con los equipos de trabajo y se les asignan objetivos a corto, mediano y largo plazo.

Mediante la metodología japonesa Kanban se:

- Define el flujo de trabajo.
- Establecen las fases del ciclo de producción.
- “Stop Starting”, “start finishing”.
- Tiene un control.

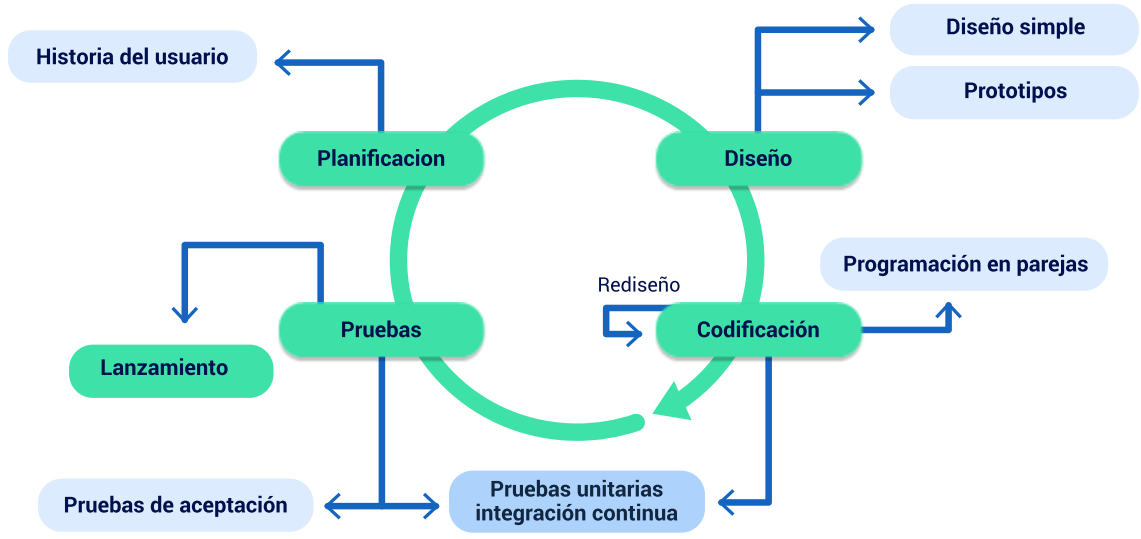
Modelo XP o programación extrema

La programación extrema o “eXtreme Programming” (XP) es un enfoque de la ingeniería de “software”, formulado por Kent Beck, autor del primer libro sobre este tema: “Extreme Programming Explained”: “Embrace Change” (1999). Esta metodología es adaptable según las necesidades y requerimientos a implementar, además, el cliente se encuentra involucrado en el proceso de desarrollo, lo que hace que el producto pueda ser terminado en un menor tiempo.

Características principales de la programación extrema:

- Tipo de desarrollo iterativo e incremental.
- Pruebas unitarias.
- Trabajo en equipo.
- Trabajo junto al cliente.
- Corrección de errores.
- Reestructuración del código.
- El código es de todos.
- El código simple es la clave.

Figura 2. Modelo XP



Nota. Adaptada de Muradas (2020).

2. Fase de definición de requisitos

En esta primera fase del ciclo de vida del "software", **también llamada fase de análisis**, se recopila, se examina y se formulan los requisitos del cliente, así como la verificación de las posibles restricciones que se puedan aplicar.

Por eso, la etapa de análisis en el ciclo de vida del "software" corresponde al proceso a través del cual se intenta descubrir qué es lo que realmente se necesita y se llega a una comprensión adecuada de los requerimientos del sistema (las características que el sistema debe poseer).

La etapa de análisis es esencial, debido a que sin esta no se sabe con precisión qué es lo que se necesita y ningún proceso de desarrollo permitirá obtenerlo. El problema que comúnmente se presenta al inicio de la etapa de desarrollo, es que el cliente no sepa exactamente que necesita, por tanto, se debe averiguar con ayuda de diferentes técnicas.

Por otra parte, la inestabilidad de los requerimientos de un sistema es inevitable, porque se estima que el 25 % de los requerimientos iniciales de un sistema, cambian antes que el sistema comience a utilizarse. Por ello, muchas prácticas resultan efectivas para gestionar adecuadamente los requerimientos de un sistema y, en cierto modo, controlar su evolución.

En la siguiente tabla, se describen las actividades y los artefactos que se realizan en la fase de definición de requisitos.

Tabla 1. Actividades y artefactos de la fase de definición de requisitos

Fase	Actividades	Artefactos
Análisis (definición de requisitos).	Definición del alcance del proyecto.	
Análisis (definición de requisitos).	Identificación del negocio.	Modelo del negocio.
Análisis (definición de requisitos).	Toma de requerimientos.	Análisis y realización de casos de uso.
Análisis (definición de requisitos).	Estudio de procesos de negocio.	Modelo de procesos y actividades de negocio.
Análisis (definición de requisitos).	Calendarización del proyecto.	Cronograma del proyecto.

3. Requisitos

Un requisito es una “condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado” (IEEE, 1990).

Los requisitos comunican las expectativas de los consumidores de productos “software”; de otra parte, los requisitos pueden ser obvios o estar ocultos, conocidos o desconocidos, esperados o inesperados, desde el punto de vista del cliente.

3.1. Importancia de los requisitos

Los requisitos cobran importancia dentro del ciclo de vida del “software”, porque:

- Establecen el alcance del trabajo subsecuente, pueden definir estrategias de desarrollo, riesgos, tomar decisiones de negocio (viabilidad de negocio), de proyecto (tiempo, recursos), de sistema (arquitectura).
- Indican al equipo del proyecto qué requieren los usuarios (necesidades de negocio).
- El éxito o fracaso de un proyecto está altamente influenciado por la calidad de los requisitos y el proceso para gestionarlos durante el desarrollo de un producto.

A continuación, se pueden revisar las características que los requisitos deben cumplir, de acuerdo con Pfleeger (2002).

- **Necesario:** si se tiene alguna duda acerca de la necesidad del requerimiento, se puede preguntar “¿Qué sería lo peor de no incluirlo?” Si no se encuentra una respuesta o cualquier consecuencia, entonces es probable que no sea un requerimiento necesario.

- **Completo:** un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- **Consistente:** un requerimiento es consistente si no es contradictorio con otro requerimiento.
- **Correcto:** acuerdo entre dos partes. Contiene una sola idea.
- **Factible:** el requerimiento deberá de ser totalmente factible y dentro de presupuesto, calendario y otras restricciones, si se tiene alguna duda de su factibilidad, hay que investigar, generar pruebas de concepto para saber su complejidad y factibilidad, si aun así el requerimiento es no factible, hay que revisar la visión del sistema y replantear el requerimiento.
- **Modificable:** los cambios en los requisitos deben hacerse de manera sistemática, y debe tenerse en cuenta su impacto en otros requisitos.
- **Priorizado:** categorizar el requerimiento nos ayuda a saber el grado de necesidad del mismo: esencial/crítico, deseado, opcional verificable.
- **Verificable:** si un requerimiento no se puede comprobar, entonces, ¿cómo se sabe si se cumplió con él o no? Debe ser posible verificarlo ya sea por inspección, análisis de prueba o demostración. Cuando se escriba un requerimiento, se deberán determinar los criterios de aceptación.
- **Rastreable:** la especificación se debe organizar de tal forma que cada función del sistema se pueda rastrear hasta su conjunto de requerimientos correspondiente. Facilita las pruebas y la validación del diseño.
- **Claro:** un requerimiento es conciso si es fácil de leer y entender, su redacción debe ser simple y clara para quienes lo consulten en un futuro.

3.2. Clasificación

Los requerimientos se pueden definir de distintas maneras, la primera clasificación se encuentra relacionada con el nivel de descripción con la que cuentan estos y dentro de este tipo de clasificación se encuentran:

- **Requerimientos de usuario**

  Son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema proporcione y de las restricciones bajo las cuales debe funcionar.

- **Requerimientos de sistema**

  Estos requerimientos establecen con detalle las funciones, servicios y restricciones operativas del sistema. El documento de requerimientos del sistema deberá ser preciso, y definir exactamente lo que se va a desarrollar.

En la siguiente clasificación se observa la que se da a los requerimientos del sistema, la cual se encuentra dividida con base en lo que se va a describir. Las clasificaciones son:

- **Requerimientos funcionales**

  Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares, o también pueden declarar explícitamente lo que el sistema no debe hacer.

- **Requerimientos no funcionales**

  Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y

estándares. Dentro de estos requerimientos se encuentra todo lo referente a la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento.

En la siguiente tabla se presentan algunos ejemplos sobre requisitos funcionales y no funcionales.

Tabla 2. Requisitos

Funcionales	No funcionales
Se debe ingresar cédula, nombre y teléfono de cada cliente.	Las consultas deben resolverse en menos de 3 segundos.
Se requiere un listado de clientes por zona.	El lenguaje de programación debe ser Java.

Se puede ampliar el tema de requisitos funcionales y no funcionales en los videos que se proponen dentro del material complementario.

4. Ingeniería de requisitos

Las siguientes son definiciones de ingeniería de requisitos de algunos autores.

“La ingeniería de requisitos es la disciplina para desarrollar una especificación completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en dónde se describen las funciones que realizará el sistema”. (Boehm, 1979).

“La ingeniería de requisitos es el proceso de estudiar las necesidades del usuario para llegar a una definición de requisitos de sistema, hardware o “software””. (IEEE, 1990).

“La ingeniería de requisitos puede considerarse como un proceso de descubrimiento y comunicación de las necesidades de clientes y usuarios y la gestión de los cambios de dichas necesidades”. (Amador, 2000).

El término IR “ingeniería de requisitos” ha surgido para englobar los procesos de desarrollo y gestión de requisitos en el ciclo de vida del “software”, el primer término (ingeniería), se enfoca en las actividades de obtención, análisis, especificación y validación de los requisitos que permitirá alcanzar los objetivos del negocio y el segundo (requisitos), está centrado en la administración de los mismos y tiene como propósito central, la gestión de los cambios y la trazabilidad, de esta forma la IR proporciona el mecanismo apropiado para:

- Entender lo que el cliente quiere.
- Analizar las necesidades.
- Evaluar la factibilidad.
- Negociar una solución razonable.

- Especificar la solución sin ambigüedades.
- Validar la especificación.
- Administrar los requisitos conforme éstos se transforman en un sistema operacional.

- **Etapas de la ingeniería de requisitos**

  Existen cuatro (4) etapas en un proceso usual de ingeniería de requisitos y que son utilizadas para el desarrollo de un producto único: elicitación, análisis, especificación y validación de los requisitos.

- **Elicitación**

  Actividad involucrada en el descubrimiento de los requisitos del sistema. Aquí los analistas deben trabajar junto con el cliente para descubrir el problema que el sistema debe resolver, los diferentes servicios que el sistema debe prestar y las restricciones que se pueden presentar.
  Los principales objetivos que se deben alcanzar son los siguientes:

  - Conocer el dominio del problema, de forma tal que los analistas puedan entenderse con los clientes y usuarios y sean capaces de transmitir dicho conocimiento al resto del equipo.
  - Descubrir necesidades reales entre clientes y usuarios, haciendo énfasis en aquellas que la mayor parte de las veces se asumen y toman por implícitas.
  - Consensuar los requisitos entre los propios clientes y usuarios hasta obtener una visión común de los mismos.

- **Análisis**

Sobre la base de la obtención realizada previamente, comienza esta fase, la cual tiene como propósito, descubrir problemas con los requisitos del sistema identificados hasta el momento, para ello se basa en los siguientes objetivos:

- Detectar conflicto en los requisitos que suelen provenir de distintas fuentes y presentar contradicciones o ambigüedades debido a su naturaleza informal.
- Profundizar en el conocimiento del dominio del problema puede facilitar el proceso de construir un producto útil para clientes y usuarios (Durán, 2000).

En esta fase, el analista proporciona un sistema de retroalimentación que refina el entendimiento conseguido en la etapa de obtención.

- **Especificación**

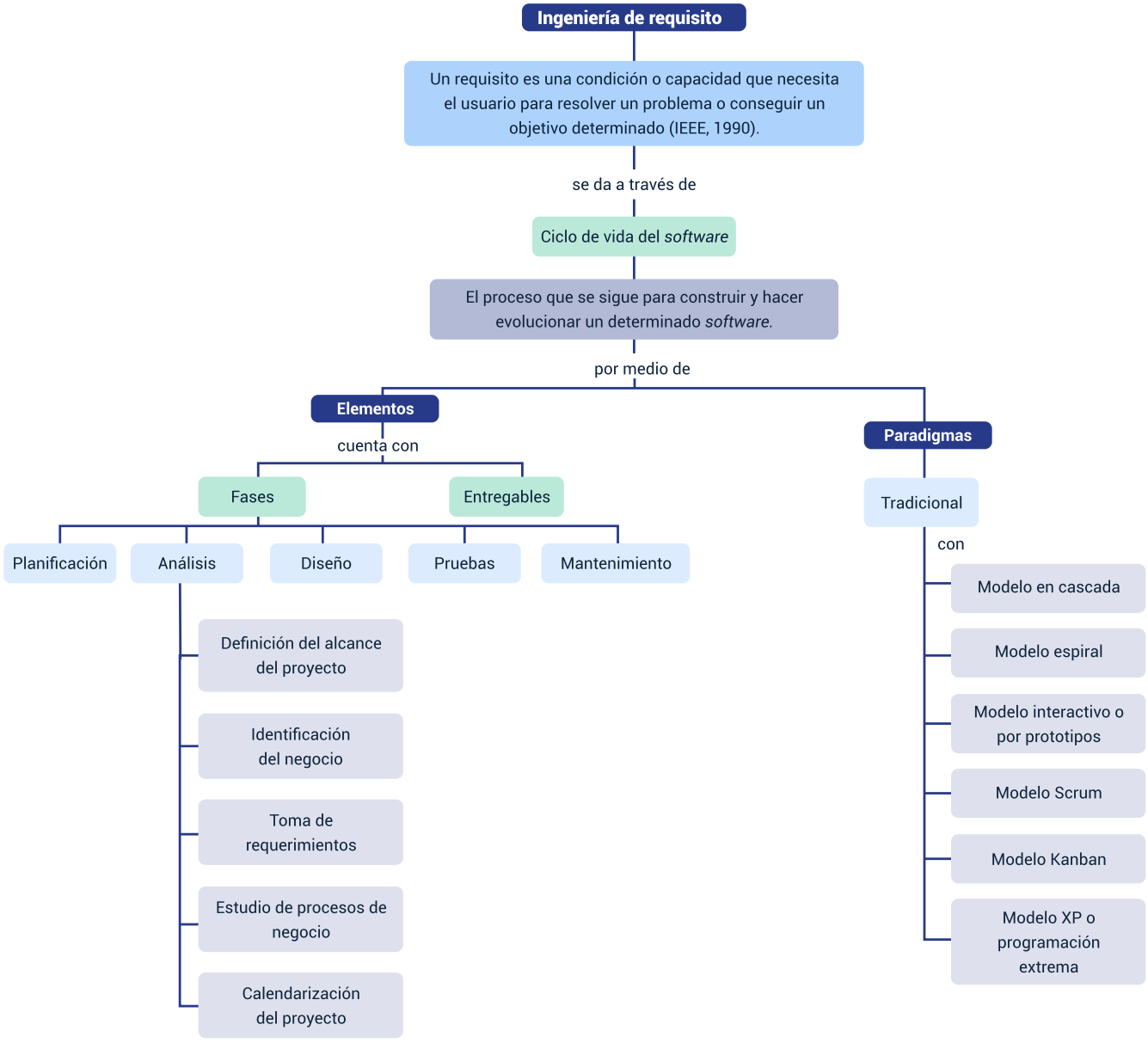
  Aquí se documentan los requisitos acordados con el cliente, en un nivel apropiado de detalle. En la práctica, esta etapa se realiza conjuntamente con el análisis, por lo que se puede decir que la especificación es el "pasar en limpio" el análisis realizado previamente aplicando técnicas y/o estándares de documentación, como la notación UML (Lenguaje de Modelado Unificado), que es un estándar para el modelado orientado a objetos, por lo que los casos de uso y la obtención de requisitos basada en los casos de uso se utilizan cada vez más para la obtención de requisitos.

- **Validación**

  Por último, la validación garantiza que los requisitos, una vez analizados y resueltos los posibles conflictos, correspondan realmente a las necesidades de clientes y usuarios, para evitar que, a pesar de que el producto final sea

técnicamente correcto, no sea satisfactorio. La validación puede llevar al analista a reescribir algunas especificaciones de requisitos y, en otros casos, a obtener nuevos, producto de la aparición de necesidades que hasta entonces estaban ocultas, para volver a evaluar el análisis inicial, o para corregir y perfeccionar el conjunto de requisitos documentados.

Síntesis


Ingeniería de requisito
Un requisito es una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado (IEEE, 1990).
se da a través de
Ciclo de vida del *software*
El proceso que se sigue para construir y hacer evolucionar un determinado *software*.
por medio de
Elementos
cuenta con
Fases
Entregables
Planificación
Análisis
Diseño
Pruebas
Mantenimiento
Definición del alcance del proyecto
Identificación del negocio
Toma de requerimientos
Estudio de procesos de negocio
Calendarización del proyecto
Paradigmas
Tradicional
con
Modelo en cascada
Modelo espiral
Modelo interactivo o por prototipos
Modelo Scrum
Modelo Kanban
Modelo XP o programación extrema

Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Ingeniería del "software": un enfoque práctico	Pressman, R. (1993). Ingeniería del "software": un enfoque práctico. McGraw-Hill Inc.	Libro	https://doku.pub/documents/ingenieria-de-software-un-enfoque-practico6thedicion-rogerpressman1-p6lkgywex804
Ingeniería del "software" - Ciclo de vida	Universidad Católica de Murcia. (2015). Ingeniería del "software" - Ciclo de vida - Raquel Martínez. YouTube.	Video	https://www.youtube.com/watch?v=4tWmULUzVdE&t=199s
Tipos de requerimientos	Itunes U – UAEH. (2019). Tipos de requerimientos. YouTube.	Video	https://www.youtube.com/watch?v=PUyfzEzSUSg
Requisitos funcionales y no funcionales	CavernaTech. (2019). Requisitos funcionales y no funcionales. YouTube.	Video	https://www.youtube.com/watch?v=Lv7XbZtnQ6A

Glosario

Ágil: comprende un conjunto de tareas o acciones que se utilizan para producir y mantener productos, así como para lograr los objetivos del proceso. La actividad incluye los procedimientos, estándares, políticas y objetivos para crear y modificar un conjunto de productos de trabajo.

Ciclo de vida de "software": aplicación de metodologías para el desarrollo del sistema "software" [AECC, 1986].

Método: indica cómo construir técnicamente el "software". Se incluyen técnicas de modelado y otras técnicas descriptivas.

Metodología: colección de métodos para resolver un tipo de problema.

Requerimiento: se refiere a la petición que se hace de algo que se solicita.

Requisito: condición que debe cumplir algo, en general el requisito cumple con lo que se requiere con el requerimiento.

Referencias bibliográficas

Boehm, B. W. (1979). “A Spiral Model of software Development and Enhancement”. ACM “Software Engineering Notes”, 11(4), 22-42.

Durán, A., y Bernárdez, B. (2001). Metodología para el análisis de requisitos de sistemas “software”.

McCracken, D., y Jackson, M. A. (1981). “A Minority Dissenting Opinion”. En W. W. Cotterman, J. D. Couger, N. L. Enger, F. Harold (Eds.). “Systems Analysis and Design: A Foundation for the 1980s” (pp. 551-553). Elsevier.

Pantaleo, G., y Rinaudo, L. (2018). Ingeniería de “software”. Alfaomega.
https://www.iso.org/obp/ui/#iso:std:iso-iec:12207:ed-2:v1:en

Penzenstadler, B. (s. f.). “Requirements Engineering”. CSU Long Beach.
http://www.foss2serve.org/index.php/Requirements_Engineering,_CSU_Long_Beach,_Penzenstadler

Pfleeger, Sh. (2002). Ingeniería del “software”. Teoría y práctica. “Prentice Hall”.

Porfirio, C. (s. f.). Técnicas de priorización: el desafío de conseguir un orden para las funcionalidades. atSistemas - Consultoría it blog.
https://www.knowmadmood.com/es/blog/tcnicas-de-priorizacin-el-desafo-de-conseguir-un-orden-para-las-funcionalidades

Rivadeneira, M., S. (2014). Metodologías ágiles enfocadas al modelado de requerimientos. Informes Científicos Técnicos - UNPA, 5(1), 1-29.
https://doi.org/10.22305/ict-unpa.v5i1.66

Sommerville I. (2011). Ingeniería del “software”. Addison-Wesley.

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Casteñanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Zulema Yidney León Escobar	Experta temática	Centro de teleinformática y producción industrial - Regional Cauca
Jonathan Guerrero Astaiza	Experto temático	Centro de teleinformática y producción industrial - Regional Cauca
Ana Catalina Córdoba Sus	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Blanca Flor Tinoco Torres	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortes	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Zuleidy María Ruiz Torres	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Luis Gabriel Urueta Alvarez	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro de Servicios de Salud - Regional Antioquia