INGENIERÍA DE SOFTWARE I
JAIME ZAMBRANA CHACÓN

CONTENIDO

Capítulo 1 INGENIERÍA DE SOFTARE 1

DEFINICIÓN DE SOFTWARE 1

DEFINICIÓN DE INGENIERÍA DE SOFTWARE 1

HISTORIA DE LA INGENIERÍA DE SOFTWARE. 2

CICLO DE VIDA DEL SOFTWARE. 5

MODELOS DEL PROCESO DE SOFTWARE. 10

CLASIFICACIÓN DE MODELOS 13

Capítulo 2 INGENIERÍA DE REQUERIMIENTOS 14

INTRODUCCIÓN 14

INTRODUCCIÓN REQUERIMIENTO 15

TIPOS DE REQUERIMIENTOS 16

EL PROCESO DE INGENIERÍA DE REQUERIMIENTOS 17

Capítulo 3 MODELADO DE SOFTWARE 22

MODELO 22

¿POR QUE EL SOFTWARE NECESITA MODELOS? 22

QUE ES UML 23

UML Y LA ORIENTACIÓN A OBJETOS 24

MODELOS EN UML 2.2 24

DIAGRAMAS 25

PILARES DE LA POO. 25

EJERCICIOS 29

Capítulo 4 gestión de proyectos de software 34

ESTIMACIÓN DE COSTOS DE DESARROLLO DE SOFTWARE 34

METODOLOGÍAS ÁGILES 37

CALENDARIZACIÓN DE PROYECTOS DE SOFTWARE 52
Bibliografía 56
Anexo A: ACTIVIDADES 57

Capítulo 1 INGENIERÍA DE SOFTARE

La ingeniería del software, según la definición de la IEEE en 1993, menciona que es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software. La ingeniería del software ofrece métodos o técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo, y trata áreas muy diversas de la informática y de las ciencias computacionales.

DEFINICIÓN DE SOFTWARE

"Programas de ordenador y la documentación asociada. Este producto denominado software se puede desarrollar para algún cliente en particular o para un mercado en general" (Sommerville, 2005, p. 5)

DEFINICIÓN DE INGENIERÍA DE SOFTWARE

Según (Sommerville, 2005), la ingeniería del software es una disciplina de ingeniería que comprende todos los aspectos de la producción de software**:**

- ¿Entonces cuál es la diferencia entre ingeniería del software y ciencia de la computación? La ciencia de la computación comprende la teoría y los fundamentos; la ingeniería del software comprende las formas prácticas para desarrollar y entregar un software útil.
- ¿Cuál es la diferencia entre ingeniería del software e ingeniería de sistemas? La ingeniería de sistemas se refiere a todos los aspectos del desarrollo de sistemas informáticos, incluyendo hardware, software e **ingeniería** de procesos. La **ingeniería** del software es parte de este proceso.

HISTORIA DE LA INGENIERÍA DE SOFTWARE.

El concepto de ingeniería del software surgió en 1968, tras una conferencia en Garmisch (Alemania) que tuvo como objetivo resolver los problemas de la crisis del software. El término crisis del software se usó desde finales de 1960 hasta mediados de 1980 para describir los frecuentes problemas que aparecían durante el proceso de desarrollo de nuevo software. Tras la aparición de nuevo hardware basado en circuitos integrados, comenzaron a desarrollarse sistemas y aplicaciones mucho más complejos que hasta entonces no era posible construir puesto que el hardware disponible no lo permitía. Estos nuevos proyectos de desarrollo de software, en la mayoría de las ocasiones, no se terminaban a tiempo, lo cual también provocaba que el presupuesto final del software excediera de aquel que se había pactado. Algunos de estos proyectos eran tan críticos (sistemas de control de aeropuertos, equipos para medicina, etc.) que sus implicaciones iban más allá de las pérdidas millonarias que causaban. Además, en muchos casos el software no daba respuesta a las verdaderas necesidades del cliente o había que ser un usuario experto para poder utilizarlo, todo ello sumado a que el mantenimiento de los productos era complejo y muy costoso.

El software no se producía como el hardware, que tenía un proceso de fabricación definido y dividido en fases. El resultado eran productos de pésima calidad en los que se habían invertido mucho tiempo y dinero pero que o bien no llegaban a terminarse o bien a la larga no daban el resultado que se esperaba. Se detectó que los métodos de desarrollo de software informales que hasta entonces habían bastado para proyectos pequeños no eran suficientes para los nuevos y grandes proyectos, y que se necesitaban profesionales especializados en esta nueva disciplina que fueran capaces de lidiar con la creciente complejidad de los nuevos sistemas.

Una de las primeras y más conocidas referencias a los conceptos crisis el software e ingeniería del software fue hecha por Edsger Dijkstra, durante la presentación de 1972 titulada “The Humble Programmer” en español “El humilde programador” en la Association for Computing Machinery, cuando se le hizo entrega de un Premio Turing.

DÉCADA DE LOS 70

Para la década de los setenta la evolución de los sistemas distribuidos, las redes de área local y global y la creciente demanda de acceso instantáneo a los datos supuso una fuerte presión sobre los desarrollos de software incrementó notablemente la complejidad de los sistemas informáticos, lo que incide en la identificación de las diferentes fases del desarrollo de software como requerimientos, análisis, codificación y pruebas, es aquí donde se introduce la programación estructurada y los métodos formales para especificar software, se identifican principios de diseño, como modularidad, encapsulación, abstracción de tipos de datos, acoplamiento débil y alta cohesión, se publica el modelo en cascada y se definen los conceptos de verificación y validación.

DÉCADA DE LOS 80

La década de los ochenta se caracteriza por la productividad y escalabilidad de sistemas y equipos de desarrollo, la industria del software es la cuna de la economía del mundo donde las técnicas para el desarrollo de software de cuarta generación (4GLs) cambian la forma en que se construyen los programas para incrementar la productividad a través de la programación por el usuario, se introducen la tecnología de programación orientada a objetos a través de múltiples lenguajes de programación desplazando los enfoques de desarrollo tradicionales.

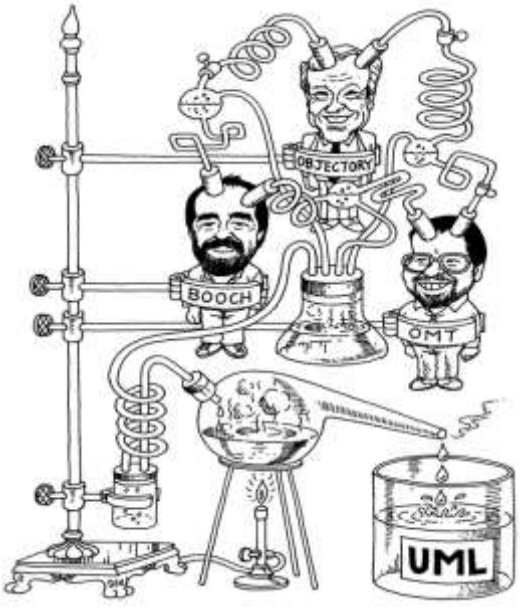


Ilustración 1 Lenguaje de Modelado Unificado (UML)

Adquiere mayor importancia, la orientación a objetos y se extiende a las fases de análisis y diseño. Las notaciones para modelar objetos como el método Booch, Objectory, OMT se unirían para crear UML, se implementa el lenguaje de modelado (UML) y se genera el primer proceso comercial de desarrollo orientado objetos (RUP).

DÉCADA DE LOS 90

En los noventa los diseñadores y los arquitectos de software inician su experiencia a través de patrones de diseño y de arquitectura. Para esta década el software era privado entonces surge la necesidad por parte de un grupo de programadores de crear proyectos que impulsan la creación de software libre y de código abierto. La usabilidad de sistemas se convierte en el foco de atención e investigación, el software empieza a ocupar la posición crítica en el mercado competitivo y en la sociedad Web. El mundo de servicios web se iría a un mundo de microservicios, y el crecimiento de infraestructuras Web apareció y pronto sería la plataforma por defecto gracias a compañías como Amazon, Google, Microsoft, IBM entre otras.

DÉCADA DE LOS 2000

metodología Scrum

En esta década nace el término **ágil**. Ken Schwaber y Jeff Sutherland adaptaron al mundo del software el termino SCRUM. La metodología Scrum es un marco de trabajo o framework que se utiliza dentro de equipos que manejan proyectos complejos. Es decir, se trata de una metodología de trabajo ágil que tiene como finalidad la entrega de valor en períodos cortos de tiempo y para ello se basa en tres pilares: la transparencia, inspección y adaptación. Esto permite al cliente, junto con su equipo comercial, insertar el producto en el mercado pronto, rápido y empezar a obtener ventas (Sales enablement).

KEN SCHWABER

Creador de la metodología Scrum

JEFF SUTHERLAND

Uno de creadores de Scrum

EXtreme Programming

KENT BECK

Creador de XP

Kent Beck, casi en la misma época lanzaría la programación extrema "eXtreme Programming". La metodología XP es una metodología ágil y flexible utilizada para la gestión de proyectos. XP, se centra en potenciar las relaciones interpersonales del equipo de desarrollo como clave del éxito mediante el trabajo en equipo, el aprendizaje continuo y el buen clima de trabajo. Esta metodología pone el énfasis en la retroalimentación continua entre cliente y el equipo de desarrollo y es idónea para proyectos con requisitos imprecisos y muy cambiantes.

DÉCADA DE LOS 2020

Actualmente estamos en un nuevo cambio, la IA está influyendo el mundo de la ingeniería de software de una forma distinta a como antes. ¡Todo ello gracias al crecimiento de la data de todas partes (imágenes, videos, audio) y el poder de la computación!, el Big Data. El futuro es incierto.

CICLO DE VIDA DEL SOFTWARE.

El ciclo de vida del desarrollo Software (SDLC en sus siglas en ingles), es una secuencia estructurada y bien definida de las etapas en Ingeniería de software para desarrollar el producto software deseado.

Una definición más precisa, propuesta por la ISO, International (Organization for Standardization), en su norma 12207 define al ciclo de vida de un software como "un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, explotación y mantenimiento de un producto software, abarcando la vida del sistema desde la definición de requisitos hasta que se deja de utilizar" (López Sanz, 2000, p. 4)

Para un conocimiento amplio de un ciclo de vida, es necesario precisar los siguientes términos.

¿Qué es un proceso?

Un proceso es un conjunto de actividades que se suceden siguiendo una ordenación temporal determinada

¿Qué es una actividad?

Una actividad es un conjunto de tareas

¿Qué es una tarea?

Es una acción que transforma unas entradas en unas salidas

FASES GENÉRICAS DEL CICLO DE VIDA DEL SOFTWARE

Fase de definición. Tareas	Ingeniería de sistemas
	Planificación del proyecto del software
	Análisis de los requisitos
Fase de desarrollo. Tareas:	Diseño del Software
	Generación de código
	Prueba del Software
Fase de mantenimiento. Cambios:	Corrección
	Adaptación
	Mejora
	Prevención

Según la norma ISO/IEC 12207-2008, las actividades que se pueden llevar a cabo durante el ciclo de vida del software se pueden agrupar en:

- 5 procesos principales
- 8 procesos de soporte
- 4 procesos de organización o generales.

En la siguiente ilustración, se puede observar las 3 agrupaciones principales denominadas como procesos del del ciclo de vida del software.

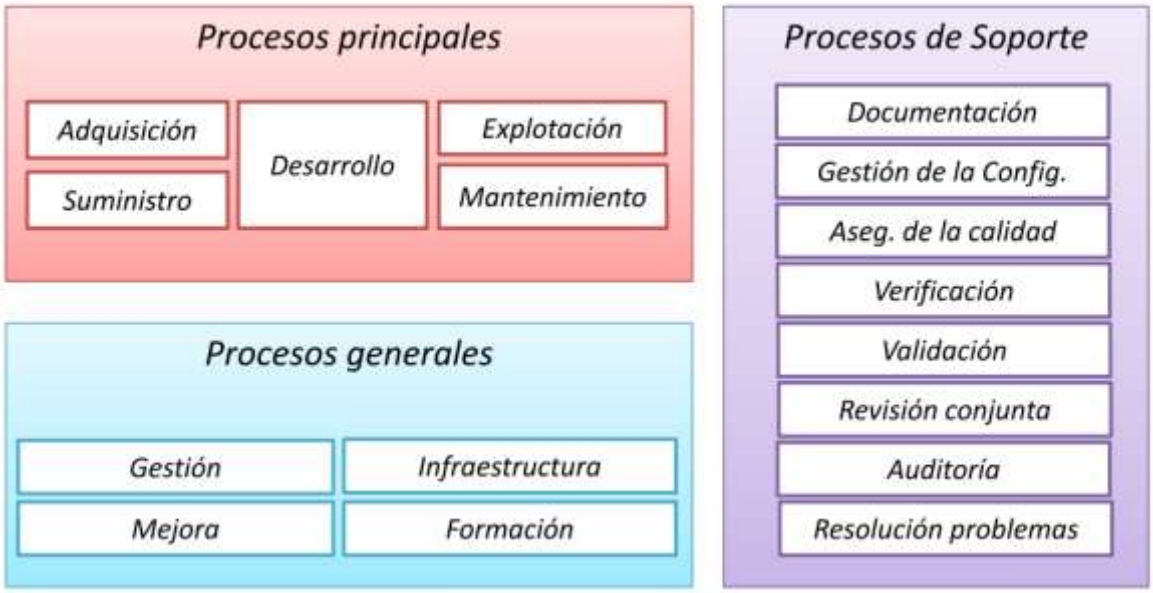


Ilustración 2 Procesos del ciclo de vida del software según la norma ISO/IEC 12207-2008

Los procesos principales tienen un componente denominado Desarrollo, que está compuesta por 6 actividades para llevar adelante el desarrollo del software que se ilustra a continuación:

Ilustración 3 proceso de desarrollo del software

Cada una de estas actividades, está compuesta por diferentes tareas. A continuación, detallaremos cada uno de estos procesos y sus actividades.

PROCESOS PRINCIPALES

Adquisición

Actividades y tareas que el comprador, el cliente o el usuario realizan para adquirir un sistema, un servicio o un producto software:

- Preparación y publicación de ofertas
- Selección del suministrador del software

Suministro

Actividades y tareas del suministrador:

- Preparar contratos como respuesta a una petición de un comprador de un producto software.
- Identificar los recursos necesarios para llevar a cabo con éxito el desarrollo del producto de software.

Desarrollo

Actividades y tareas enfocadas a la obtención de un producto Software:

- Análisis
- Diseño
- Codificación
- Pruebas
- Integración
- Implantación

Explotación

Explotación del software y soporte operativo a los usuarios.

Mantenimiento

Actividades que incluyen modificaciones del producto, tanto del código como de la documentación, debido a errores o a la necesidad de mejora o adaptación:

- Migración hacia un nuevo entorno operativo.
- Retirada del producto.

PROCESOS DE SOPORTE

Estos procesos, dan soporte al resto de procesos y se aplican durante cualquier momento del ciclo de vida del software.

Documentación

Registra la información producida por un proceso o actividad del ciclo de vida, además de diseñar, editar, distribuir y mantener los documentos producidos durante el desarrollo del software.

Gestión de la configuración

Son las actividades que controlan las modificaciones y versiones de los elementos como el registro de las peticiones de cambios e informar de los estados de estos.

Aseguramiento de la calidad

Son actividades para asegurar que los productos cumplen los requisitos especificados y se ajustan a los planes establecidos.

Verificación

Actividades para determinar el buen funcionamiento de un producto software

Validación

Son las actividades para determinar si el producto cumple los requisitos previstos

Revisión conjunta

Actividades que permiten determinar el estado de los productos en una determinada actividad del ciclo de vida o en una cierta fase del proyecto. Puede ser una reunión conjunta con el cliente, el grupo de desarrollo y los clientes potenciales para revisar el trabajo hecho.

Auditorías

Actividades que permiten determinar en unos momentos determinados si se han conseguido los objetivos propuestos: requisitos, cumplimiento de contrato, etc.

Resolución de problemas

Actividades que permiten analizar y resolver los problemas o disconformidades con los requisitos o con el contrato, que hayan surgido durante el desarrollo, la explotación, el mantenimiento, o en cualquier otro momento. Además de disponer de un medio documental que permita asegurar que todos los problemas ya se han tratado.

PROCESOS GENERALES

Estos procesos de soporte a la organización contribuyen a la formación del personal y mejora los procesos, a continuación, se describen:

Gestión

Actividades de planificación, seguimiento, control, revisión y evaluación.

Infraestructura

Actividades para determinar la infraestructura necesaria para un proceso. Incluye hardware y software, instalaciones, etc.

Mejora

Permite valorar, medir, controlar, evaluar y mejorar todos los procesos del ciclo de vida.

Formación

Permite la formación continua de los empleados en conocimientos, habilidades y aptitudes.

MODELOS DEL PROCESO DE SOFTWARE.

La primera pregunta que un ingeniero en software debe de realizarse es:

Entonces se debe tener las siguientes consideraciones:

- Representación abstracta de un proceso del software
- También se considera como estrategias de desarrollo que ayudan a organizar las diferentes etapas y actividades del ciclo de vida del software con los modelos de ciclo de vida del software.
- Estos modelos ayudan al control y a la coordinación del proyecto
- El modelo para utilizar depende del tipo de proyecto.

MODELO EN CASCADA

El primer modelo de proceso de desarrollo de software que se publicó se derivó de procesos de ingeniería de sistemas más generales (Royce, 1970). (Sommerville, 2005)

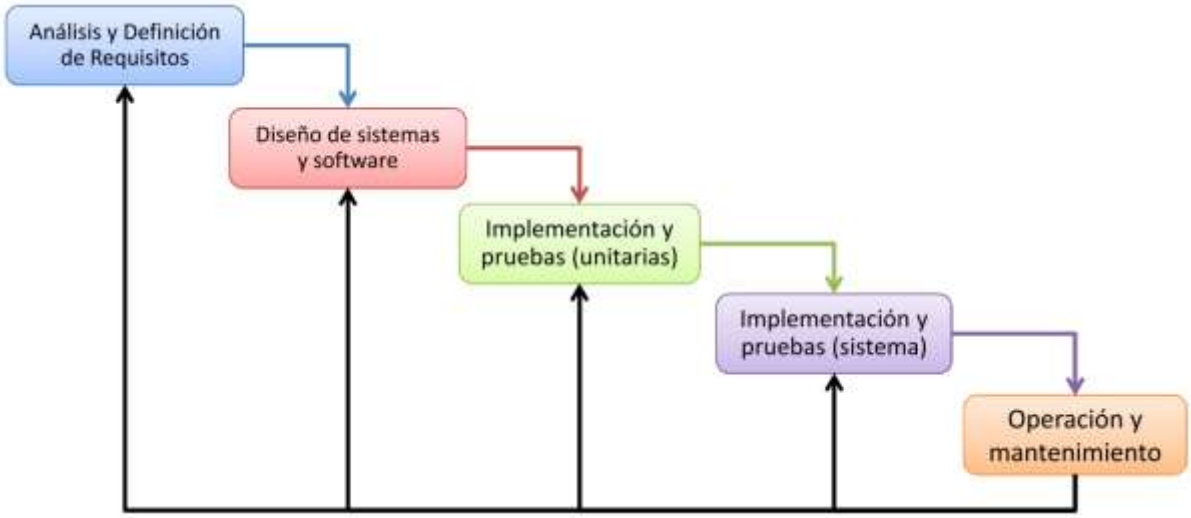
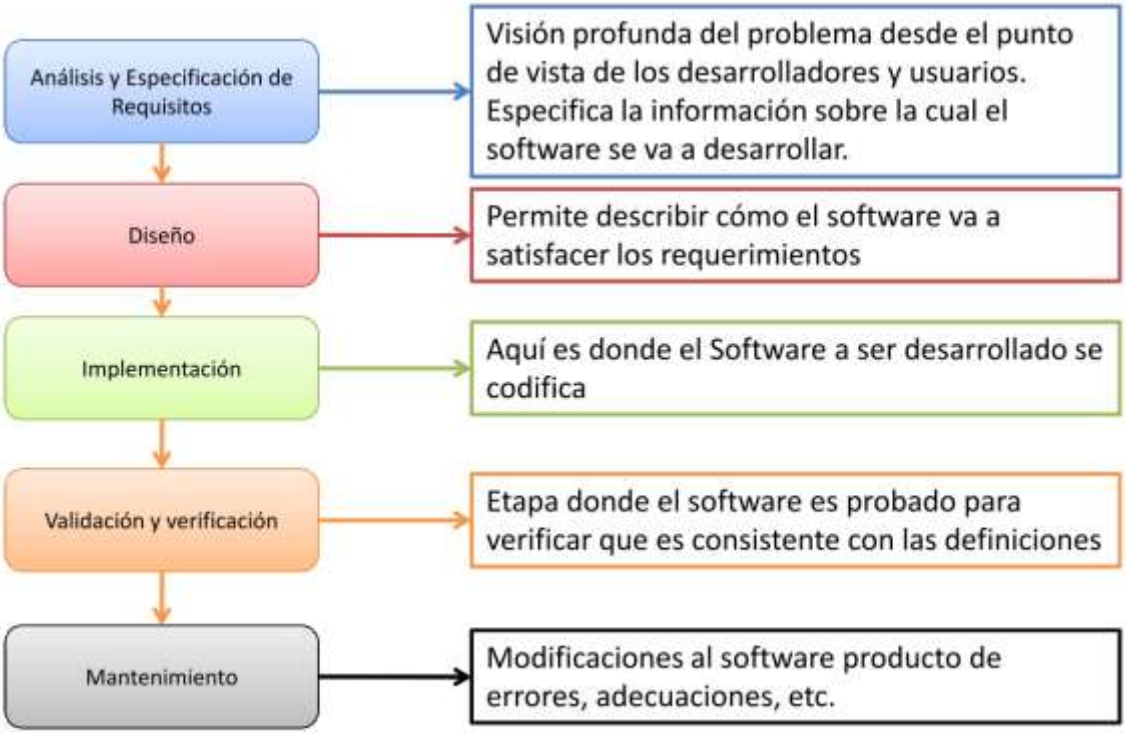


Ilustración 4 Modelado en Cascada



MODELO DE PROTOTIPOS

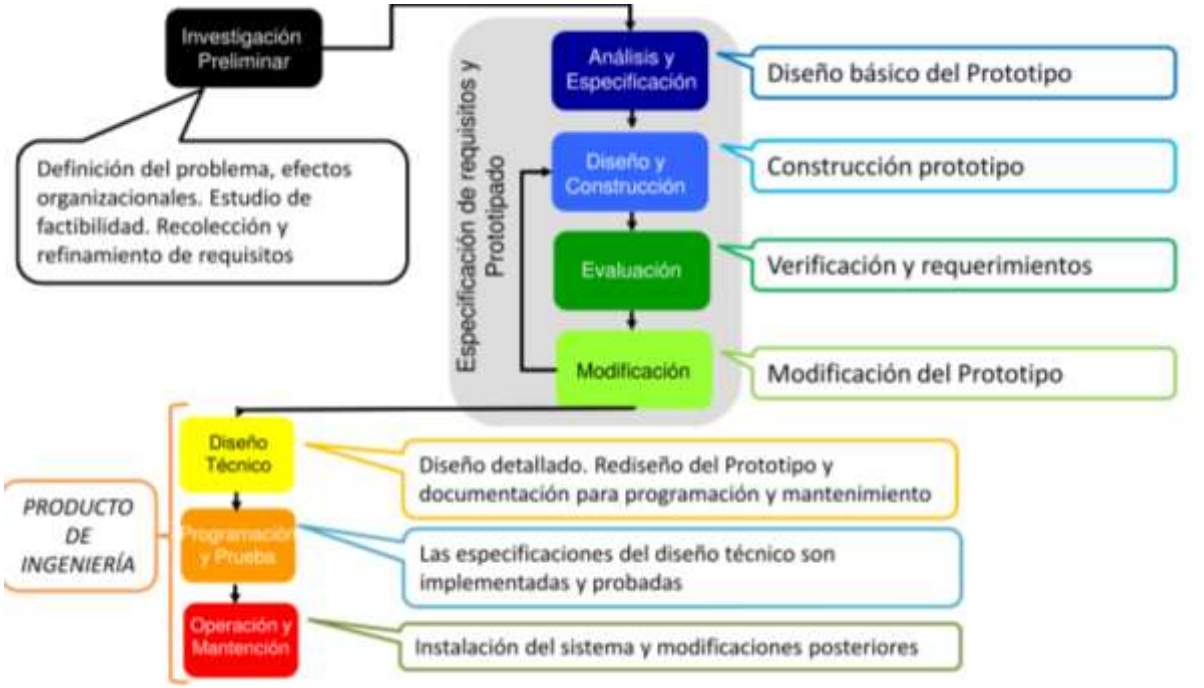


Ilustración 5 Modela de prototipos

MODELO EN ESPIRAL

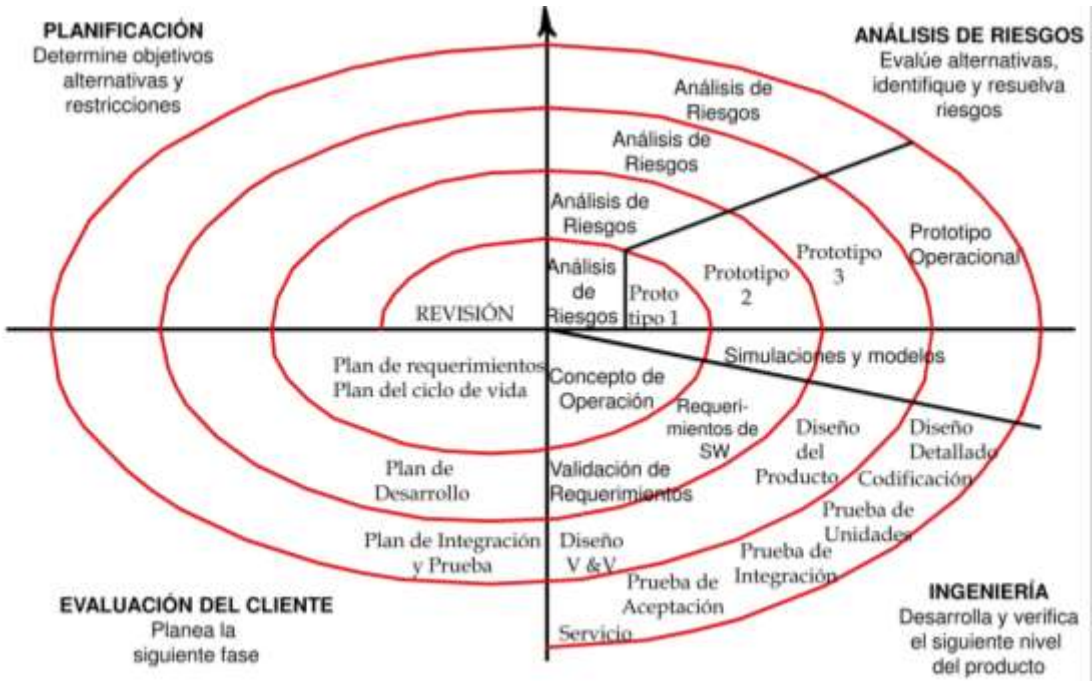


Ilustración 6 Modelado en Espiral

CLASIFICACIÓN DE MODELOS

Existen muchos modelos que tienen la siguiente clasificación (ver ilustración 7) de los modelos y metodologías concretos más citadas en la literatura que son considerados como modelos abstractos: Cascada, Evolutivos, Minimización de Desarrollos, Híbridos y Ágiles. Esto se divide en Modelos Tradicionales (también llamados pesados), que son los que promueven la disciplina por medio de la planificación y la comunicación escrita, y los Metodologías Ágiles, que dan prioridad a la interacción entre los individuos y a la comunicación con el cliente.

Modelo Abstracto		Modelos Concretos
Tradicionales o Pesados	En Cascada	Pura Con fases solapadas Con subproyectos Con reducción de riesgos
	Evolutivos	Espiral Entrega por etapas o incremental Entrega evolutiva o iterativo Diseño por planificación Cascada en V
	Minimización de Desarrollos	Componentes Reutilizables Diseño por herramientas
	Híbridos	Proceso Unificado Racional Otros
Metodologías Ágiles		Programación extrema SCRUM Desarrollo dirigido por pruebas Desarrollo dirigido por Características Agile, Lean, Crystal, …, etc.

Ilustración 7 Clasificación de modelos concretos en clases abstractas.

Las metodologías ágiles se explicarán en el capítulo 4.

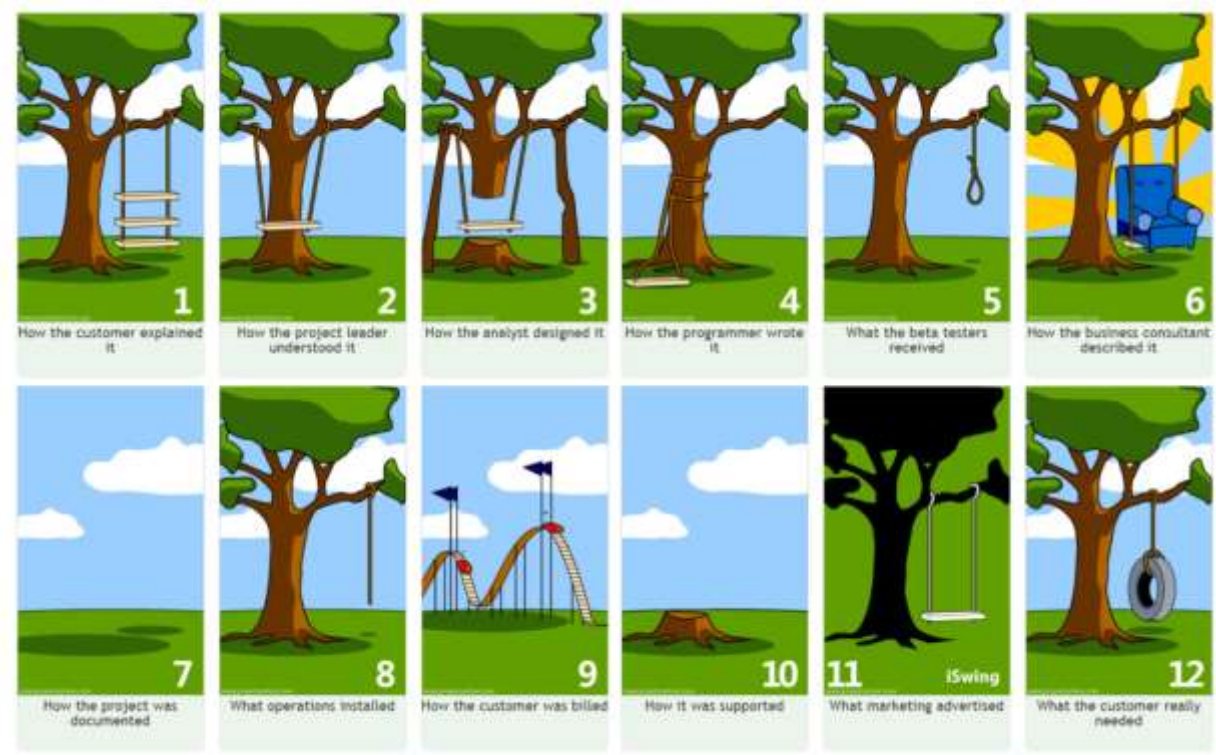
Capítulo 2 INGENIERÍA DE REQUERIMIENTOS

INTRODUCCIÓN

En la actualidad, son muchos los procesos de desarrollo de software que existen. Con el pasar de los años, la Ingeniería de Software ha introducido y popularizado una serie de estándares para medir y certificar la calidad, tanto del sistema a desarrollar, como del proceso de desarrollo en sí. Se han publicado muchos libros y artículos relacionados con este tema, con el modelado de procesos del negocio y la reingeniería. Un número creciente de herramientas automatizadas han surgido para ayudar a definir y aplicar un proceso de desarrollo de software efectivo. Hoy en día la economía global depende más de sistemas automatizados que en épocas pasadas; esto ha llevado a los equipos de desarrollo a enfrentarse con nuevos procesos y estándares de calidad.

La Ingeniería de Requerimientos cumple un papel primordial en el proceso de producción de software, ya que enfoca un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas.

Estudios realizados muestran que más del 53% de los proyectos de software fracasan por no realizar un estudio previo de requisitos. Otros factores como falta de participación del usuario, requerimientos incompletos y el cambio a los requerimientos, también ocupan sitiales altos en los motivos de fracasos.



INTRODUCCIÓN REQUERIMIENTO

La Ingeniería de requerimientos (IR) es considerara como “el proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software es llamado Ingeniería de Requerimientos”. La meta de la ingeniería de requerimientos es entregar una especificación de requerimientos de software correcta y completa. La ingeniería de requerimientos apunta a mejorar la forma en que comprendemos y

definimos sistemas de software complejos. Existen varias definiciones de requerimientos, de entre las cuales podemos citar las siguientes:

De acuerdo con la definición de la IEEE, un requerimiento es:

- Una condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo (IEEE).
- Una condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta (IEEE).

De acuerdo con otros autores como (Robertson - Robertson) definen requerimiento de software como algo que el sistema debe hacer o una cualidad que el sistema debe poseer.

Según Zave:

- Rama de la ingeniería del software que trata con el establecimiento de los objetivos, funciones y restricciones de los sistemas software.
- Asimismo, se ocupa de la relación entre estos factores con el objeto de establecer especificaciones precisas.

Según Boehm:

- Ingeniería de Requerimientos 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.

Así que se tienen muchas definiciones, pero en este documento, nos quedamos con la primera.

El proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software es llamado Ingeniería de Requerimientos.

TIPOS DE REQUERIMIENTOS

Al revisar los requerimientos de software, podemos identificar varios tipos:

REQUERIMIENTOS FUNCIONALES:

Funcionalidades del software.

REQUERIMIENTOS NO FUNCIONALES

Exigencias en otros aspectos del software. Por ejemplo, requerimientos en torno a atributos de calidad como rendimiento, tolerancia a fallas y seguridad.

REGLAS DE NEGOCIO

Validaciones, reglas y normas que deben ser soportadas por el software.

OTROS REQUERIMIENTOS:

Otros tipos de exigencias que no pertenecen a las otras categorías. Por ejemplo, tiempos, personal involucrado y presupuestos para el proyecto.

EL PROCESO DE INGENIERÍA DE REQUERIMIENTOS

Es el proceso consiste en cuatro actividades (no secuenciales) según (Rcasalla, 2020) así:

- Recopilar, descubrir (Elicitation)
- Analizar
- Documentar (Especificar)
- Validar (Lograr un acuerdo con el cliente)

Los pasos anteriores no definen un proceso secuencial**.** Se trata de un proceso iterativo donde el orden depende de los avances y del entendimiento que se logre con el cliente.

Gradualmente durante estos pasos se va construyendo un documento de definición de requerimientos.

RECOPILAR, DESCUBRIR (ELICITATION)

Es la obtención y el descubrimiento de los requerimientos del software según diversos Stakeholders (constituyentes) y otras fuentes (leyes, restricciones). Las técnicas para realizar esta actividad son las entrevistas, el análisis de documentos, los grupos de discusión, los cuestionarios.

Siempre debemos buscar eliminar las ambigüedades que llevan a interpretaciones distintas del mismo requerimiento. Los problemas comunes son:

- Requerimientos olvidados en particular los no funcionales y las restricciones
- Palabras ambiguas (“pequeño”, “barato”, ...)
- ¡Vocabulario dependiente del contexto del negocio!
- Creer que se entendió (el ingeniero)
- Creer que se explicó claro (cliente)
- Pasar por alto lo importante
- Lo necesario vs. lo ideal
- Lo actual vs. el cambio

Un proceso posible para identificar los requerimientos es:

Enunciar el problema	**El problema puede ser definido como una diferencia entre cómo las cosas son percibidas y cómo son deseadas.**
Enunciar el propósito	Identificar lo que daría más valor al cliente.
Modelar el mundo del problema y definir un vocabulario común.	Identificar los conceptos y las relaciones importantes en los elementos del mundo del problema. Por ejemplo, utilizando un Diagrama de clases UML.
	Explicar los conceptos (clases) y las relaciones en un glosario.
Entrevistar	Validar el entendimiento con el cliente. Obtener más información.

ANÁLISIS DE REQUERIMIENTOS

Analizar los requerimientos significa:

- Determinar si los requerimientos son los indicados: claros, completos, coherentes, si se podrán probar (testable)
- Resolver los conflictos que puedan surgir entre los involucrados (stakeholders)

Para llevar adelante el análisis, se necesitan técnicas que permitan abstraer, modelar, identificar funcionalidad, identificar restricciones, características, prototipar, simular, discutir, …

Casos de uso

Los casos de uso se utilizan para especificar los escenarios en los que un actor interactúa con la aplicación. Básicamente permite documentar, con mucho más detalle, requerimientos funcionales de la aplicación considerando los diferentes escenarios y reglas de negocio que deben implementarse.

Es posible especificar los requerimientos funcionales usando casos de uso de una manera gradual. La idea es construir una especificación inicial e irla completando a medida que se avanza en el análisis de los requerimientos. Una especificación mínima puede incluir una pequeña descripción e información de las entradas y salidas.

DOCUMENTACIÓN DE LOS REQUERIMIENTOS

A continuación, se presenta un ejemplo, de cómo documentar los requerimientos.

Documento de Especificación de Requerimientos

Versión	**1.0 Inicial Básica**
Fecha	Marzo 15 de 2021
Autor	TecnoProfe

Contexto

La empresa Mundo Hogar, quiere ofrecer a sus usuarios una tienda virtual para sus productos, libros, ropa, cuadernos, esferos, etc. Quiere contar con un catálogo de la librería en línea donde se pueda hacer búsquedas bajo distintos criterios y adquirir los productos por internet.

Alcance

Este sistema se ocupará solo de la venta de los libros.

Glosario de Términos

Concepto	Descripción
Book	Libro en la tienda.
Author	Autor de uno o muchos libros disponibles en la tienda. Un libro puede tener varios autores.
Editorial	representa las empresas Editoriales de los libros. Cada libro tiene una editorial.

Requerimientos Funcionales

Actores

Para este sistema hay dos actores: el usuario Comprador y el Administrador. El Administrador es quien se ocupará de crear la información en el sistema.

Casos de Uso

	Nombre	Resumen
CU1	Consultar datos de un libro	El sistema permite que un usuario consulte la información de un libro en particular
CU2	Crear un libro	El sistema permite el registro de un libro nuevo en la tienda
CU2	Ver la información de todos los libros	EL sistema muestra el catálogo de libros de la librería.

Reglas de negocio

- No existen dos libros con el mismo ISBN
- No se puede eliminar un autor si tiene algún libro registrado

Requerimientos No Funcionales

Algunos requerimientos no funcionales, relacionados con atributos de calidad son:

- El software debe soportar varios usuarios trabajando simultáneamente
- La interfaz usuario debe ser muy intuitiva y agradable.

Otras Restricciones

Entre los otros requerimientos, están

- El software deberá funcionar en servidores con Sistema Operativo GNU/LINUX.
- Servidores de bases de datos MySQL
- La administración de versiones debe hacerse en Gitlab.

VALIDACIÓN DE LOS REQUERIMIENTOS

El propósito de validar los requerimientos de software con el Cliente es corroboran que el entendimiento sobre lo que se espera que haga el sistema es correcto. De nuevo, buscamos evitar construir el sistema que no se necesita.

Las técnicas para validar los requerimientos pueden ser basadas en:

- Inspección de los documentos por parte del cliente y como resultado él reportará sus hallazgos.
- Presentaciones y reuniones y como resultado se discute y argumenta y se trata de llegar a un acuerdo.
- Prototipos en donde el cliente puede ver más fácilmente lo que será la aplicación.
- Una mezcla de las anteriores (esta siempre es la mejor opción).

Capítulo 3 MODELADO DE SOFTWARE

MODELO

UML es el acrónimo en inglés para "lenguaje unificado de modelado" Pero ¿qué significa modelar? La respuesta corta seria: “construir modelos”, Sin embargo, esta respuesta nos lleva a otra pregunta: ¿qué es un modelo?

La palabra modelo tiene varias acepciones en castellano. Para nosotros, en este libro, y en el contexto de UML, un modelo "es una descripción analógica para ayudar a visualizar algo que no se puede observar directamente y que se realiza con un propósito determinado y se destina a un público especifico”.

¿POR QUE EL SOFTWARE NECESITA MODELOS?

El software necesita modelos por las mismas razones que cualquier otra construcción humana para comunicar de manera sencilla una idea abstracta, existente o no, o para describir un producto existente.

En efecto, el modelo más detallado de un producto de software es el código fuete. Pero es como decir que el mejor modelo de un edificio es el edificio mismo; esto no nos sirve para concebirlo antes de la construcción ni para entender sus aspectos más ocultos con vistas al mantenimiento.

Sin embargo, en el software el modelado es aún más importante que en las otras ingenierías Esto tiene varias razones de ser:

- El software es invisible e intangible: solo se ve su comportamiento, sus efectos en el medio.

- El software es mucho más modificable que otros productos realizados por el hombre esta modificabilidad es percibida por los ajenos a la industria, lo que provoca que haya un incentivo mucho más fuerte para pedir modificaciones.
- El software se desarrolla por proyectos, no en forma repetitiva como los productos de la industria manufacturera, Esto hace que cada vez que construyamos un producto de software estemos enfrentándonos a un problema nuevo.
- El software es sustancialmente complejo, con cientos o miles de partes interactuando, diferentes entre sí y que pueden ir cambiando de estados a lo largo de su vida. esto hace que analizar un producto de software requiera mecanismos de abstracción y de un lenguaje para representarlo.
- El desarrollo del software es inherentemente complejo, la complejidad del producto lleva a la complejidad de los proyectos, de los equipos de desarrollo y de la administración de proyectos.

QUE ES UML

UML es una notación de modelado visual, que utiliza diagramas para mostrar distintos aspectos de un sistema. Si bien muchos destacan que UML es apto para modelar cualquier sistema, su mayor difusión y sus principales virtudes se advierten en el campo de los sistemas de software. Esto no obsta para que muchos profesionales intenten usar UML en situaciones diversas, haciendo uso de esa máxima que dice que "cuando la única herramienta que conocemos es el martillo, aun los tomillos nos parecen clavos”.

UML, Surgió en 1995, por iniciativa de Grady Booch, James Rumbaugh e ivar Jacobson, tres conocidos ingenieros de software que ya habían avanzado con sus propias notaciones de modelado. Precisamente, UML se define como "unificado", porque surgió como síntesis de los mejores elementos de las notaciones previas.

Y nos ha venido muy bien, ya que, a mediados de la década de 1990, nos encontrábamos empantanados en la falta de un estándar, aunque fuese de facto, que marcase el camino para la modelización de software orientado a objetos.

Luego UML se especificó con más rigurosidad y en 1997, se presentó la versión 1.0, que fue aprobada y establecida como estándar por el OMG (Object Management Group. un consorcio de empresas de desarrollo de estándares). De allí en más, sigue

evolucionando, formalizándose y lo que no siempre es una ventaja creciendo y complejizándose.

Hacia 2000, UML ya se habla convertido en el estándar de facto para modelización de software orientado a objetos.

En la actualidad, UML es un lenguaje de visualización, especificación y documentación de software. basado en trece tipos de diagramas, cada uno con sus objetivos.

UML Y LA ORIENTACIÓN A OBJETOS

La palabra **unificado** dentro del acrónimo UML ha llevado a muchos a tratar de usar UML para modelar cualquier tipo de software, independientemente del paradigma de desarrollo.

Lo cierto es que todo puede hacerse. Sin embargo, no hay que olvidar que UML surgió en el marco del paradigma orientado a objetos, por lo que se aplica más naturalmente a ellos.

MODELOS EN UML 2.2

UML trabaja con 13 tipos de diagramas.

Los diagramas estructurales

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de objetos (estático).
- Diagrama de paquetes
- Diagrama de componentes.
- Diagrama de despliegue.
- Diagrama de estructuras compuestas.

Los diagramas de comportamiento

- Diagrama de secuencia.
- Diagrama de comunicación (o de colaboración).
- Diagrama de máquina de estados o de estados.
- Diagrama de actividades.
- Diagrama de visión global de la interacción.

- Diagrama de tiempos.

DIAGRAMAS

Diagrama de clases.

Los diagramas de clases forman la columna vertebral arquitectónica de muchos modelos de sistemas procesos. Un diagrama de clase consta de los siguientes elementos principales:

- Clase
- Atributo
- Método
- Asociación y composición.

> Los diagramas de clases describen la estructura de un sistema de software y así forme la primera notación de la base discutida para el modelado orientado a objetos.

Clase

Una clase consta de una colección de atributos y métodos que determinan el estado y el comportamiento de sus instancias (objetos). Las clases están conectadas entre sí por asociaciones y relaciones hereditarias. El nombre de la clase identifica a la clase.

Atributo

Los componentes de estado de atributo de una clase se denominan atributos. Se describe un atributo por su nombre y tipo.

Método

La funcionalidad de una clase se almacena en métodos y describe la implementación.

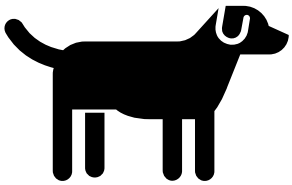
PILARES DE LA POO.

La POO tiene varios pilares para asegurar la simplicidad de código y su reutilización, y aunque diversos autores señalan diversos pilares, en este documento

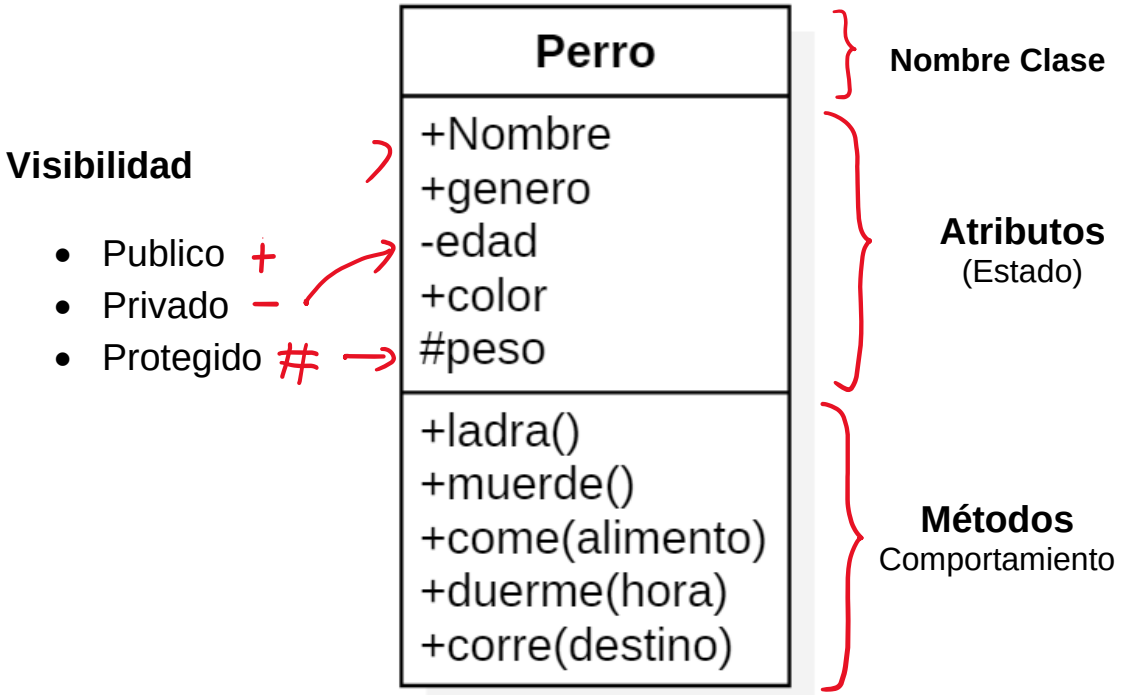
se considerarán los cuatro que son comunes en la mayoría de los textos, estos son: Abstracción, Encapsulamiento, Herencia y Polimorfismo, las dos primeras están más relacionadas con la búsqueda de códigos simples y las dos siguientes con la reutilización.

Abstracción

Es el pilar de la POO, que permite identificar las características y comportamientos de un objeto y con los cuales se construirá la clase (plantilla). Esto quiere decir que a través de este pilar o fundamento es posible reconocer los atributos y métodos de un objeto.



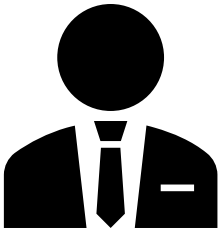
Si se toma en cuenta a un perro como objeto, se puede apreciar que tiene: nombre, genero, edad, color, peso, etc. Estas son características generales que se obtiene o mejor dicho abstraen de este bello animalito. A ese proceso de obtener esos elementos, se le llama abstracción, porque cualquier perrito, puede tomar valores similares que le identifiquen.



De igual forma, se identifican los comportamientos o que puede tener un perro.

Encapsulamiento

Es la característica de la POO que permite el ocultamiento de la complejidad del código, pertenece a la parte privada de la clase y que no puede ser vista desde ningún otro programa.



Docente
+nombres +apellidos +fecha_nacimiento +edad +sexo +usuario +password +sueldo
+trabaja(horas) +imparteclase(asignatura) +poner_nota() +vacaciones(tiempo)

En el gráfico se representa a una persona que trabaja en una empresa privada, a esta persona no le gustaría revelar el sueldo que gana, puesto que está reservado para su empleador y los responsables de recursos humanos. En la empresa se le

facilito un usuario y contraseña, por supuesto la contraseña solamente la debe conocer el funcionario y ninguna otra persona incluso

En realidad, el encapsulamiento está relacionado con el acceso a un código desde el código de otra clase; sin embargo, en términos generales, esta representación gráfica es conveniente para comprender el concepto de encapsulamiento.

Herencia

Es el pilar más fuerte que asegura la reutilización de código, ya que a partir de esta característica es posible reutilizar (heredar) las características y comportamientos de una clase superior llamada clase padre, a sus clases hijas, denominadas clases derivadas. Esto implica que una vez desarrollado el código de una clase base, su código puede ser reutilizado por las clases derivadas.

En el gráfico, Persona es la clase Padre, que tiene como características: CI, nombre, dirección, fechaNac, genero, entre otros; y Estudiante y Profesor son las clases "Hijas", que heredan las características de la clase padre y a su vez establecen las propias de su clase. Esto implica que no se deberán volver a definir, sino que por el simple hecho de heredarlas ya es posible utilizarlas y en el caso de los comportamientos ejecutarlos o modificarlos si es necesario.

Polimorfismo

A través de esta característica es posible definir varios métodos o comportamientos de un objeto bajo un mismo nombre, de forma tal que es posible modificar los parámetros del método, o reescribir su funcionamiento, o incrementar más funcionalidades a un método.

En el gráfico se observa que todas son figuras geométricas por lo que pueden incluirse en una clase Padre, por lo que la clase deberá tener el método Área (), este método podrá ser reescrito tantas veces como figuras existan, con los parámetros correspondientes en cada clase derivada: Circulo, Triangulo y Rectángulo, o reescrita en la clase base.

EJERCICIOS

Ejercicio 1

Se quiere hacer el diseño de un robot modular. El robot estará compuesto por varios módulos entre los que se encuentran: rotación, extensión, helicoidal, cámara. Los módulos podrán ser dinámicos (capaces de moverse: rotación, extensión, helicoidal) o estáticos (no se pueden mover: cámara).

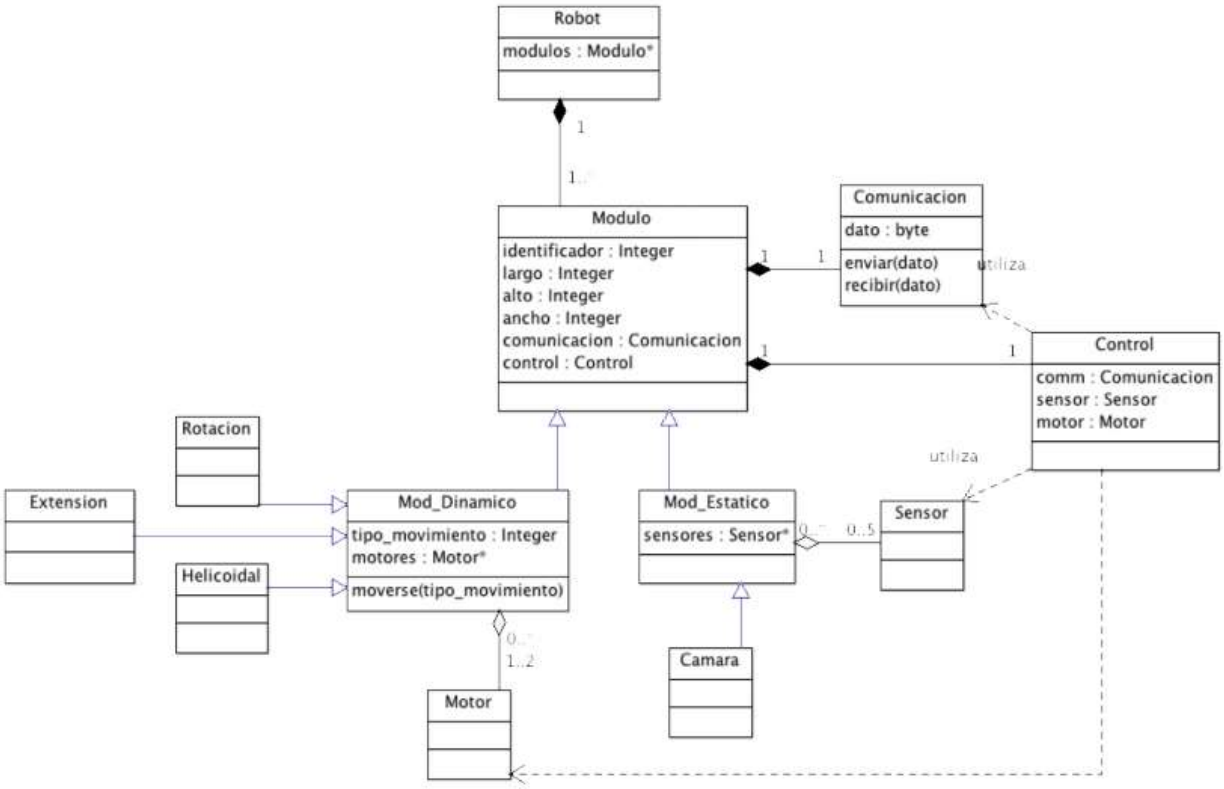
Los módulos tendrán un identificador (1-255) y unas dimensiones (largo, ancho y alto, entre 1 y 200mm). Los módulos estarán compuestos de un sistema de control y un sistema de comunicación. Los módulos dinámicos tendrán:

- Motores (1 o 2).
- Un parámetro que es el tipo de movimiento que pueden realizar.
- Una función que es moverse (con parámetro el tipo de movimiento).

Los módulos estáticos podrán tener sensores (de 0 a 5). El sistema de control utiliza el sistema de mensajes para comunicarse. Los módulos pueden enviar y recibir mensajes de/hacia el usuario y otros módulos, con un parámetro que es un array de datos a mandar o recibir. También utiliza los motores para moverse y los sensores para captar información del medio.

Se pide que utilizando herencia siempre que se pueda, se realice un diseño UML de las clases necesarias para representar todas las entidades del sistema, indicando atributos y métodos, así como las relaciones existentes entre las clases.

Solución



Ejercicio 2

Especificar un diagrama de clases que describa redes de ordenadores. Los elementos que se pueden incluir en la red son:

- Servidor, PC, Impresora.
- Hub, Cable de red.

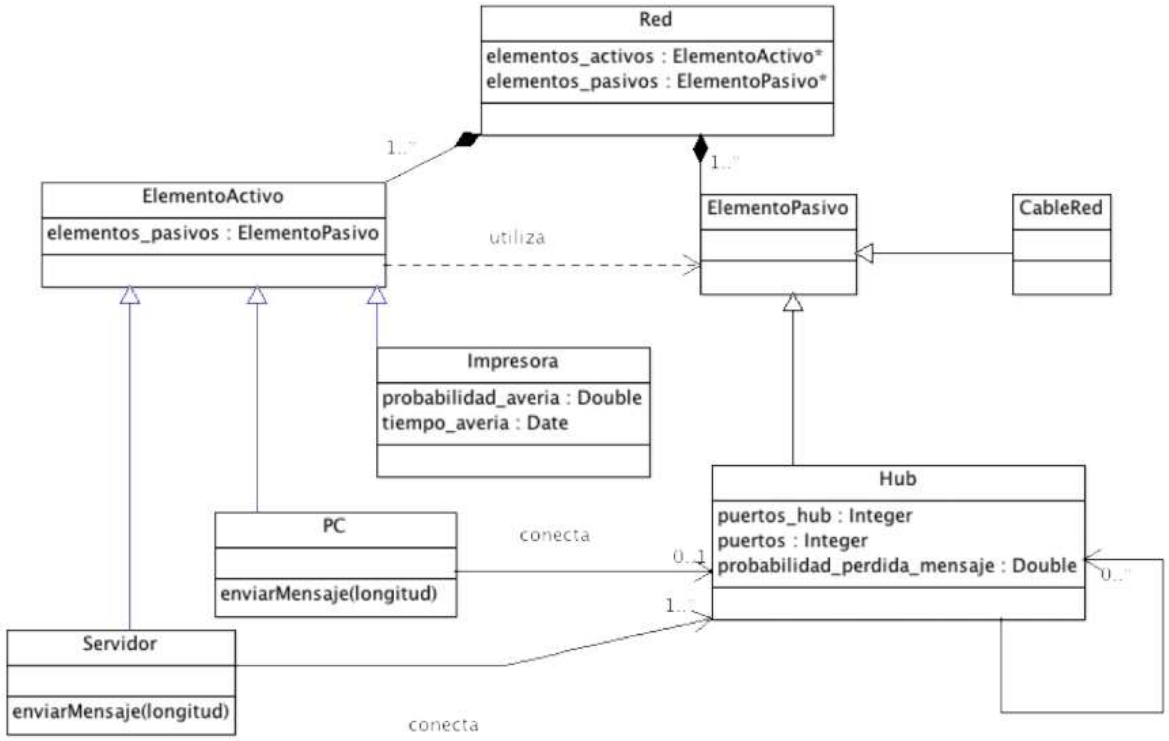
Los computadores pueden conectarse con un único Hub, los servidores con uno o varios.

Los Servidores y computadores pueden generar mensajes, con una cierta longitud.

Los Hubs tienen un número de puertos, algunos de los cuales puede usarse para conectar con otros Hubs. Tienen cierta probabilidad de “perder” mensajes.

Las impresoras pueden averiarse, con cierta probabilidad, durante cierto tiempo

Solución



Ejercicio 3

Representa mediante un diagrama de clases la siguiente especificación.

Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes. Ambos (empleados y sus clientes) se caracterizan por su nombre y edad. Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.

De los clientes, además se necesita conocer su teléfono de contacto. La aplicación necesita mostrar los datos de empleados y clientes.

Solución

Ejercicio 4

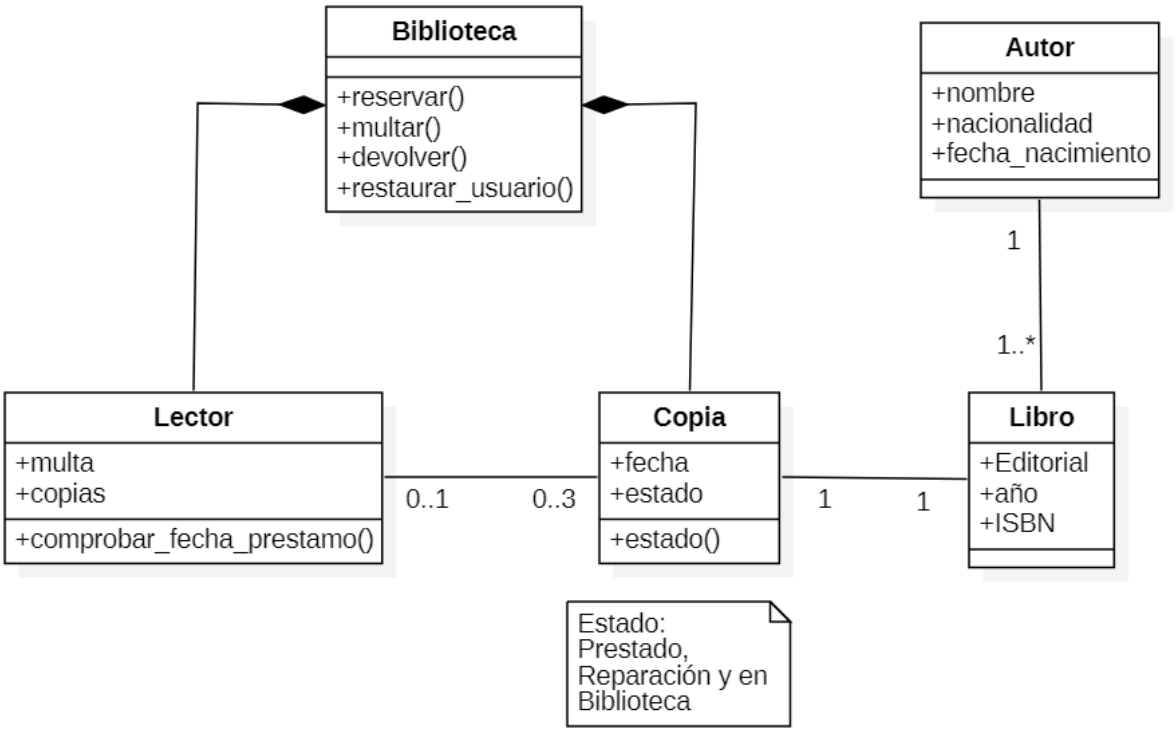
Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (ingeniería, literatura, informática, historia ...), editorial, año y autor. Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.

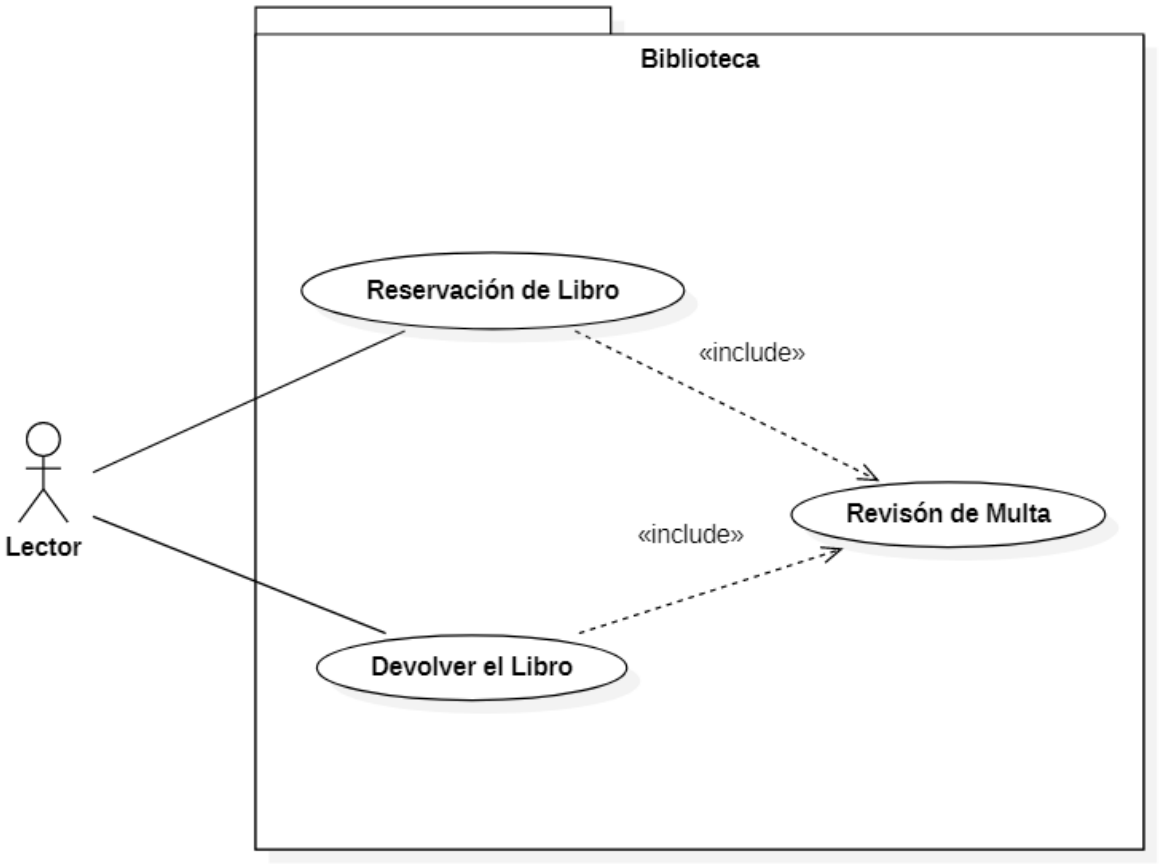
Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en reparación. Los lectores pueden tener un máximo de 3 libros en préstamo.

Cada libro se presta un máximo de 30 días, y por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un nuevo libro.

Realiza un diagrama de clases y añade los métodos necesarios para realizar el préstamo y devolución de libros. Realiza un diagrama de casos de usos.

Solución




Biblioteca
Reservación de Libro
«include»
Revisón de Multa
«include»
Lector
Devolver el Libro

Capítulo 4 gestión de proyectos de software

ESTIMACIÓN DE COSTOS DE DESARROLLO DE SOFTWARE

La estimación de los costos de desarrollo de software es un factor muy importante en el análisis de los proyectos informáticos, constituye un tema estratégico contar con indicadores para medir el costo de estos, garantizando la eficiencia, excelencia, calidad y la competitividad.

> El análisis de costo es el proceso de identificación de los recursos necesarios para llevar a cabo el trabajo o proyecto eficientemente.

El desarrollo de software es una actividad muy compleja obteniendo como resultado un producto intangible que depende principalmente del esfuerzo intelectual y creatividad de personas que lo realizan. Los errores humanos están presentes en todas las etapas de un proyecto de este tipo y puede llegar a ser muy costosa su corrección.

El resultado de un producto de software con el costo lo más exacto posible implica la utilización de metodologías, procedimientos, métricas y estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo (Jones, 1996).

Técnicas de estimación de costos en el desarrollo de software

Se han desarrollado varias técnicas de estimación para el desarrollo de software estableciendo de antemano el ámbito del proyecto, usar las métricas del

software sirven de base para la realización de estimaciones y desglosar el proyecto en partes más pequeñas que se estiman individualmente (Angelis L & Stamelo, 2000).

Debido al crecimiento en la industria del software y siendo una actividad compleja la estimación de costos, las compañías dedicadas a ofrecer diferentes herramientas de estimación de costos en el mercado han aumentado. A partir del 2005, algunas de esas herramientas de estimación son:

- COCOMO II
- CoCots
- CoStar
- CostModeler
- CostXpert
- SoftCost.

A continuación, se estudiará una de las métricas mas usadas en este campo.

COCOMO

El Modelo COCOMO Creado por Barry Boehm en 1981. Su nombre significa COnstructive COst MOdel (Modelo constructivo de costo) constituye una jerarquía de modelos de estimación para el software. La jerarquía está constituida por los siguientes modelos según el manual de COCOMO (1990).

BARRY BOEHM (1990-2002)

Científico de la computación

COCOMO básico.

Calcula el esfuerzo y el costo del desarrollo en función del tamaño del programa estimado en LOC.

COCOMO intermedio.

Calcula el esfuerzo del desarrollo en función del tamaño del programa y un conjunto de conductores de costo que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.

COCOMO detallado.

Incorpora las características de la versión intermedia y lleva a cabo una evaluación del impacto de los conductores de costo en cada fase (análisis, desarrollo, etc.) del proceso.

COCOMO es una herramienta basada en las líneas de código la cual le hace muy poderoso para la estimación de costos y no como otros que solamente miden el esfuerzo en base al tamaño. Es necesario para un administrador de proyectos una herramienta de estimación de costos; y esta herramienta puede estar relacionada con COCOMO ya que esta técnica representa uno de los más completos modelos empíricos para la estimación de software (Bumett, 1998).

Una de las deficiencias detectadas en el modelo COCOMO es que para el análisis del costo del proyecto solo analizan el salario del desarrollador sin tener en cuenta otros elementos de gastos que inciden en los costos del proyecto de software.

COCOMO II

COCOMO II es un modelo que permite estimar el costo, el esfuerzo y el tiempo cuando se planifica una nueva actividad de desarrollo software, y está asociado a los ciclos de vida modernos. Fue desarrollado a partir de COCOMO, incluyendo actualizaciones y nuevas extensiones más adecuadas a los requerimientos de los ingenieros software (Heemstra, 1992).

Está construido para satisfacer aquellas necesidades expresadas por los estimadores de software, como por ejemplo el apoyo a la planificación de proyectos, la previsión de personal del proyecto, replanificación, seguimiento, negociaciones de contrato y la evaluación del diseño.

Los factores de costo describen aspectos relacionados con la naturaleza del producto, hardware utilizado, personal involucrado, y características propias del proyecto. El conjunto de factores de escala explica los ahorros y pérdidas producidos a medida que un proyecto de software incrementa su tamaño. El usar un modelo u otro depende del nivel de detalle del proyecto, de la fidelidad requerida de las estimaciones, de la definición de los requerimientos y de los detalles de la arquitectura (Gause & Weinberg, 1989).

Capítulo 4

METODOLOGÍAS ÁGILES

METODOLOGÍA XP

KENT BECK

Creadores de XP

Extrema o eXtreme Programming (XP) es una metodología de desarrollo de la ingeniería de software, que nace oficialmente en aproximados de marzo de 1996 en un proyecto desarrollado por Kent Beck.

«Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.» (Beck, 2000)

La programación extrema es una metodología de desarrollo ligera (o ágil) basada en una serie de valores y de prácticas de buenas maneras que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas.

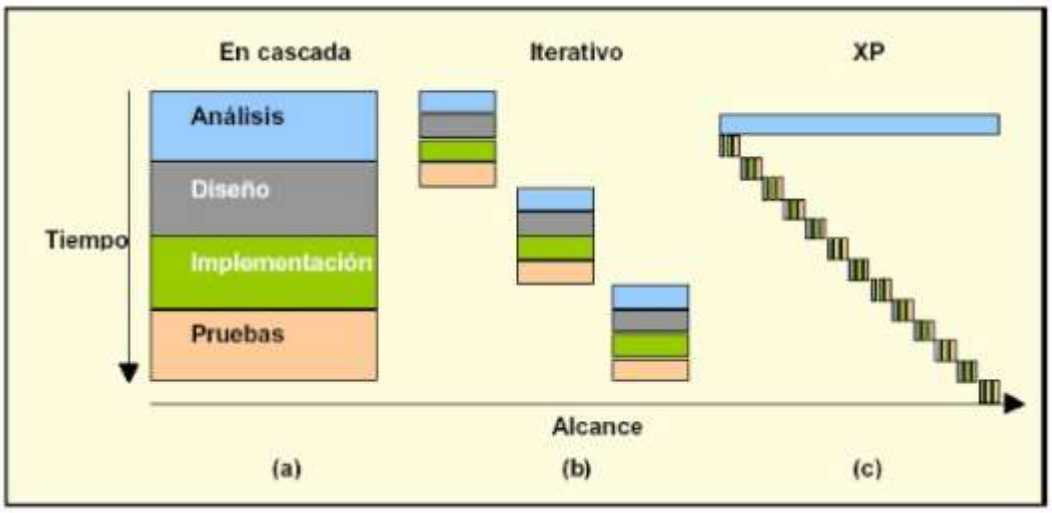
CICLO DE VIDA

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente.

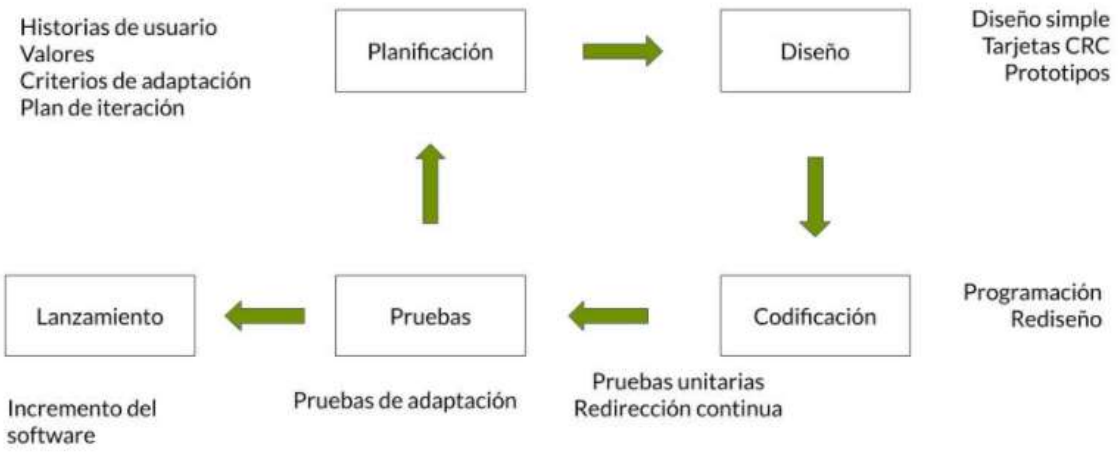
Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un proyecto. Por esto, se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo.

En cada iteración se realiza un ciclo completo de análisis, diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas que caracterizan a XP (y

que serán detalladas más adelante). Típicamente un proyecto con XP lleva 10 a 15 ciclos o iteraciones.



FASES EN XP



FASE I. PLANIFICACIÓN DEL PROYECTO

Historias de usuario

El primer paso de cualquier proyecto que siga la metodología XP es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso, pero con algunas diferencias: Constan de 3 o 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando

llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas.

Release Planning (plan de publicaciones)

Después de tener ya definidas las historias de usuario es necesario crear un plan de publicaciones, en inglés “Release plan”, donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones. Un “Release plan” es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que serán implementadas y las historias que serán implementadas en cada versión del programa. Después de un “Release plan” tienen que estar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las historias que se deben desarrollar en cada versión), el tiempo que tardarán en desarrollarse y publicarse las versiones del programa, el número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado.

Iteraciones

Todo proyecto que siga la metodología X.P. se ha de dividir en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada iteración los clientes deben seleccionar las historias de usuario definidas en el “Release planning” que serán implementadas. También se seleccionan las historias de usuario que no pasaron la prueba de aceptación que se realizó al terminar la iteración anterior. Estas historias de usuario son divididas en tareas de entre 1 y 3 días de duración que se asignarán a los programadores.

La Velocidad del Proyecto

Es una medida que representa la rapidez con la que se desarrolla el proyecto; estimarla es muy sencillo, basta con contar el número de historias de usuario que se pueden implementar en una iteración; de esta forma, se sabrá el cupo de historias que se pueden desarrollar en las distintas iteraciones. Usando la velocidad del proyecto controlaremos que todas las tareas se puedan desarrollar en el tiempo del que dispone la iteración. Es conveniente reevaluar esta medida cada 3 o 4 iteraciones

y si se aprecia que no es adecuada hay que negociar con el cliente un nuevo “Release Plan”.

Programación en Parejas

La metodología XP aconseja la programación en parejas pues incrementa la productividad y la calidad del software desarrollado. El trabajo en pareja involucra a dos programadores trabajando en el mismo equipo; mientras uno codifica haciendo hincapié en la calidad de la función o método que está implementando, el otro analiza si ese método o función es adecuado y está bien diseñado. De esta forma se consigue un código y diseño con gran calidad.

Reuniones Diarias

Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo debe tener voz y voto.

FASE II. DISEÑO

Diseños Simples:

La metodología XP sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e implementable que a la larga costará menos tiempo y esfuerzo desarrollar.

Glosarios de Términos

Usar glosarios de términos y una correcta especificación de los nombres de métodos y clases ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código.

Riesgos:

Si surgen problemas potenciales durante el diseño, XP sugiere utilizar una pareja de desarrolladores para que investiguen reduzcan al máximo el riesgo que supone ese problema.

Funcionabilidad extra:

Nunca se debe añadir funcionalidad extra al programa, aunque se piense que en un futuro será utilizada. Sólo el 10% de la misma es utilizada, lo que implica que el desarrollo de funcionalidad extra es un desperdicio de tiempo y recursos.

Refactorizar:

Refactorizar es mejorar y modificar la estructura y codificación de códigos ya creados sin alterar su funcionalidad. Refactorizar supone revisar de nuevo estos códigos para procurar optimizar su funcionamiento. Es muy común rehusar códigos ya creados que contienen funcionalidades que no serán usadas y diseños obsoletos.

Fase III. Codificación

El cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de XP. A la hora de codificar una historia de usuario su presencia es aún más necesaria. No olvidemos que los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen las pruebas que verifiquen que la historia implementada cumple la funcionalidad especificada. La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

FASE IV. PRUEBAS

Uno de los pilares de la metodología XP es el uso de prueba para comprobar el funcionamiento de los códigos que vayamos implementando. El uso de las pruebas en XP es el siguiente:

- Se deben crear las aplicaciones que realizarán las pruebas con un entorno de desarrollo específico para cada prueba.
- Hay que someter a test las distintas clases del sistema omitiendo los métodos más triviales.
- Se deben crear las pruebas que pasarán los códigos antes de implementarlos; en el apartado anterior se explicó la importancia de crear antes las pruebas que el código.
- Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará.

- Como se comentó anteriormente las distintas pruebas, se deben subir al repositorio de código acompañados del código que verifican.
- Test de aceptación. Las pruebas mencionadas anteriormente sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario.
- Al ser las distintas funcionalidades de nuestra aplicación no demasiado extensas, no se harán test que analicen partes de estas, sino que las pruebas se realizarán para las funcionalidades generales que debe cumplir el programa especificado en la descripción de requisitos

FASE V. LANZAMIENTO

El lanzamiento es sin duda el momento más esperado. Si se han seguido de forma correcta las etapas anteriores, no deberíamos hallar sorpresas.

Se supone que hemos probado todas las historias de usuario, ajustándonos a los requerimientos del cliente, por ende, se ha logrado estructurar un software que cumple con las expectativas, que ha superado las pruebas del tester y del resto del equipo.

Las cuatro fases y sus actividades más importantes de XP, se encuentra resumida en la siguiente gráfica:

Capítulo 4

ROLES DE XP

Cliente

El cliente es el responsable de conducir el proyecto. Define el proyecto y sus objetivos. Cuanto más preciso es su trabajo y cuanto mayor sea su involucración, mayores serán las oportunidades de éxito.

Programador

Una vez que se han comprendido las historias de usuario, el XP adjudica a los programadores la responsabilidad de tomar decisiones técnicas. Los desarrolladores estiman el tiempo que les va a tomar cada historia.

Encargado de Pruebas (Tester).

El encargado de pruebas ayuda al cliente a definir y escribir las pruebas de aceptación de las historias de usuario. Este rol en un equipo XP también es responsable realizar test periódicamente y informar de los resultados al equipo. A medida que el volumen de pruebas aumenta, el encargado de pruebas necesitará una herramienta para crear y mantener la batería de pruebas, ejecutarlas y obtener los resultados más rápidamente.

Encargado de Seguimiento (Tracker)

Hace el seguimiento de acuerdo con la planificación. La métrica más importante para XP es la velocidad del equipo, que se define como el tiempo ideal estimado para las tareas frente al tiempo real dedicado. Esta métrica ayuda a determinar si el proyecto está dentro del tiempo de la iteración.

Entrenador (Coach)

No es un rol cubierto en todos los equipos de XP. Su papel es guiar y orientar al equipo, especialmente cuando un equipo comienza a trabajar siguiendo la metodología XP. Esto se debe a que no es fácil aplicar XP de forma consistente. Aunque son prácticas de sentido común, se necesita un tiempo para interiorizarlas. También hay situaciones especiales en las que se requiere la sabiduría de un especialista en XP para aplicar sus normas frente a un obstáculo en el proyecto.

Gestor (Big Boss)

Es el gerente del proyecto, debe tener una idea general del proyecto y estar familiarizado con su estado. El cliente puede asumir este papel.

METODOLOGÍA SCRUM

Creadores de SCRUM

La metodología Scrum fue definida por Ikujiro Nonaka y Hirotaka Takeuchi a principios de los 80. Lo hicieron en un estudio en el que analizaron la manera de desarrollar nuevos productos de empresas tecnológicas como Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M o Hewlett-Packard.

En 1993 se realizó el primer Scrum para desarrollo de software y en 1995 el proceso fue formalizado. En 2001 un grupo de personas muy relevantes en lo que empezaba a ser el desarrollo ágil escribieron los valores fundamentales de los procesos ágiles.

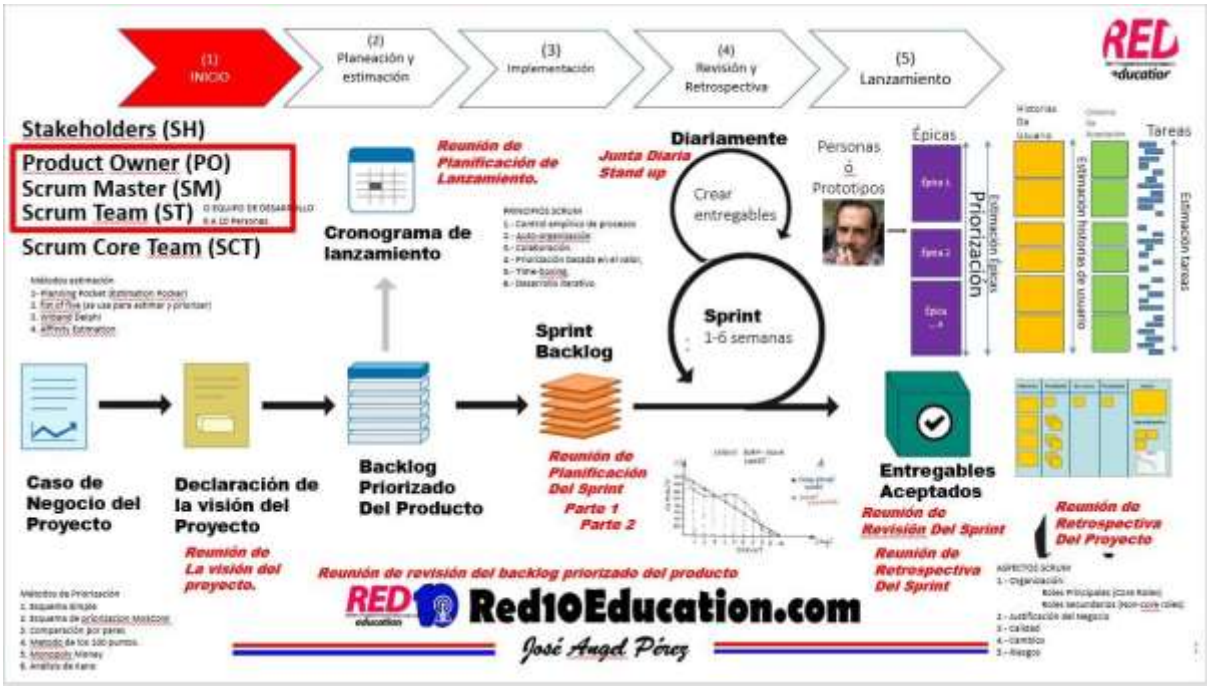
Desde 1995 miles de proyectos en todo el mundo han utilizado Scrum para el desarrollo de productos, tanto en empresas pequeñas, “startups” con tan sólo 3 personas desarrollando un producto, como en multinacionales, entre las que se encuentran las siguientes:

Sectores	**Ejemplos de empresas que utilizan metodologías ágiles como Scrum**
Media y Telcos	BBC, BellSouth, British Telecom, DoubleYou, Motorola, Nokia, Palm, Qualcomm, Schibsted, Sony/Ericsson, Telefonica I+D, TeleAtlas, Verizon
Software, Hardware	Adobe, Autentia, Biko2, Central Desktop, Citrix, Gailén, IBM, Intel, Microfocus, Microsoft, Novell, OpenView Labs, Plain Concepts, Primavera, Proyectalis, Softhouse, Valtech, VersionOne.
Internet	Amazon, Google, mySpace, Yahoo
ERP	SAP
Banca e Inversión	Bank of America, Barclays Global Investors, Key Bank, Merrill Lynch
Sanidad y Salud	Patientkeeper, Philips Medical

Defensa y Aeroespacial	Boeing, General Dynamics, Lockheed Martin
Juegos	Blizzard, High Moon Studios, Crytek, Ubisoft, Electronic Arts
Otros	3M, Bose, GE, UOC, Ferrari

En la actualidad, Scrum se está utilizando en diferentes tipos de negocio y, especialmente, en el desarrollo de software. La Scrum Alliance es la organización sin ánimo de lucro que se encarga de difundir Scrum en este ámbito. (Proyectosagiles, 2021).

Así, Scrum es un modo de trabajar en equipo en el que el resultado se produce de forma incremental. Para lograrlo se establecen periodos cortos de trabajo en los que se sigue un mismo patrón. Y se ha demostrado que es aplicable a todo tipo de empresas, no sólo las de raíz tecnológica.



FASES

Este es un resumen y está basado en la imagen anterior, tiene todas las fases y procesos de SCRUM con las herramientas, pero en una forma descriptiva.

Para fines de la explicación de SCRUM se realizó las siguientes abreviaciones:

- Stakeholder - SH.
- Product Owner - PO
- SCRUM Master - SM

- SCRUM Team - ST
- SCRUM Core Team - SCT

FASE I. INICIO

Business Project Meeting

La metodología inicia con la fase de INICIO, y la primera actividad es la reunión de la visión del proyecto (Business Project Meeting) donde a partir de un caso de negocio existente obtenemos la declaración de la visión del proyecto que es un listado a alto nivel sobre los requerimientos del proyecto, también en esta fase los Stakeholders (SH) identifican el Product Owner (PO) que será el primer integrante del Scrum Core Team (SCT), el SCT está conformado por el PO, el SM y el ST.

El PO es seleccionado por los SH porque es el más cercano al cliente, en terminos generales es la voz del cliente y es el que tiene el conocimiento del negocio dentro del SCT y por lo tanto hace la justificación del negocio.

selección del Scrum Máster e identificación de los Stakeholders

También en la fase de inicio el Product (PO) selecciona al Scrum Master (SM) e identifica a los Stakeholders (SH).

El SM es un líder servicial, es el mentor de SCRUM es el que tiene más conocimiento de la metodología y es el que va a estar encargado de eliminar interrupciones y todo lo que no permita que el SCT no cumpla sus objetivos.

Los SH son también llamados los interesados y como su nombre lo indica tienen un interés en el proyecto, tienen un impacto ya sea positivo o negativo, pueden ser, el cliente, usuarios, patrocinadores, etc.

Equipo de desarrollo

Ahora el PO ya sea solo o con ayuda del SM selecciona al SCRUM Team (ST), también llamado muchas veces equipo de desarrollo, aquí podemos mencionar uno de los Aspectos de SCRUM que es la organización (Grupos auto-organizados).

El ST es el encargado de hacer el trabajo, son los que conocen la solución a nivel de detalle (bajo nivel), es la parte técnica del desarrollo del producto.

Épicas

Mediante reuniones de grupos de usuarios y con base en la visión del proyecto se desarrollan en forma muy general las funcionalidades que debe tener nuestro producto o proyecto, las cuales se llaman Épicas, el desarrollo de las épicas se apoya en personas o prototipos que nos muestran un modelo de los requerimientos de los usuarios que usarán nuestro producto una vez terminado, todo aquello conforma lo que en un futuro será el

Backlog Priorizado del producto

Que es la lista de todas la épicas pero priorizadas por el PO en colaboración (Principio Scrum) con el ST y el cliente de tal manera que el máximo valor sea entregado como prioridad (Principio Priorizacion basada en valor) y será la base para los incrementos a desarrollar en cada Sprint o Iteración (principio de Scrum). Los sprints son divisiones en tiempo en el que partiremos nuestro proyecto, con cada sprint tendremos un incremento, al concluir los sprints cada incremento producirá el producto terminado que pueda ser comercializado por cumplir los principios de un Producto Mínimo Viable (Puede ser utilizado, aunque no esté completado), se recomienda que el sprint tenga una duración de 1 a 6 semanas y una vez establecida la longitud del Sprint no se puede cambiar.

Estimación de las épicas

Ahora el equipo de desarrollo estima las épicas con el adecuado método de estimación (ver métodos de estimación), esta estimación se realiza con base en la complejidad, esfuerzo y riesgos envueltos en el desarrollo, con lo cual se obtiene el esfuerzo estimado necesario para desarrollar las épicas (Puntos de historia) que nos indican la velocidad de trabajo en el desarrollo.

Reunión de planificación de lanzamiento

Una vez contando con el backlog priorizado del producto, puntos de historia y velocidad se tiene una reunión llamada reunión de planificación de lanzamiento (release planning meeting) “Sin time-box” donde SHs y SCT con ayuda de las estimaciones dividen el proyecto en sprints, determina la duración del sprint, la cantidad de sprints, y las fechas de terminación del producto en incrementos potencialmente entregables.

Time-box es un principio de SCRUM donde a varios eventos están restingidos en tiempo, tienen un tiempo máximo que por ejemplo hablando de reuniones evita que perdamos tiempo en otros temas.

FASE 2. PLANEACION Y ESTIMACION.

Trabajar con los requerimientos

Ahora es tiempo de trabajar sin SHs, ya con los requerimientos en el backlog priorizado del producto, el SCT empieza a detallar este backlog en el evento llamado Reunión de Planificación del Sprint (Spring planning meeting) que debe durar 2 horas por semana de duración del sprint (Time-boxed).

Presentación de las épicas, divididas en épicas más detalladas

En esta junta el product owner presenta las épicas, divididas en épicas más detalladas y refinadas que se llaman "historias de usuario" con un criterio de aceptación para definir que cumple con las expectativas de entrega, se aclaran objetivos y se aclaran dudas sobre estas historias.

Planificación del Sprint

En la reunión de planificación del Sprint también el SM facilita que ST estime las historias de usuario en base a la complejidad, esfuerzo y riesgos envueltos en el desarrollo, por ser un grupo auto-organizado el scrum team compromete las historias de usuario asignándole a cada miembro del equipo una tarea en base a su experiencia, sin olvidar que las historias de usuario son responsabilidad de todo el grupo.

Subdivisión de historias estimadas en tareas

En la reunión de planificación el scrum team subdivide las historias estimadas en tareas específicas identificadas y a desarrollar,

Estimación de las tareas

Se estiman las tareas, se les da tiempos en horas-hombre y decide las tareas que se realizarán el sprint que inicia, esta nueva lista se llama Sprint backlog.

Creación del Sprint Burndown

Se crea el Sprint Burndown chart y se inicia el #scrumboard.

El Sprint Burndown Chart nos indica el trabajo que falta por completarse.

El #Scrumboard es un tablero que nos hace transparente todas las actividades y los avances del Sprint en proceso.

FASE 3. IMPLEMENTACION.

Sprint Daily

Por fin continuamos con la implementación, todos los días antes de iniciar el día de trabajo, el sprint se inicia con una junta facilitada por el SM llamada Sprint Daily Stand up con un tiempo máximo de 15 min (time-boxed) donde de pie el scrum team responde 3 preguntas ¿Qué hice en la reunión pasada? ¿Qué pienso hacer antes de la sig. reunión? ¿Qué impedimentos tengo (si los hubiera) para hacer mi trabajo?

Alimentación del Scrumboard

Se alimenta el Scrumboard donde aparece todo lo pendiente y lo realizado y una lista de impedimentos que tiene que limpiar el Scrum Master.

Se actualiza la gráfica llamada Burndown Chart donde aparece el avance día a día del proyecto y las desviaciones en su ejecución y nos indica si se está cumpliendo el objetivo.

Participación del Product Owner en reuniones de revisión.

Durante el transcurso del sprint el product owner participa en juntas llamadas Reunión de revisión del backlog priorizado del producto donde refina el backlog priorizado del producto y lo presenta el junta de planeación del próximo Sprint.

FASE 4. REVISION Y RETROSPECTIVA.

Junta de revision del sprint (Sprint Review meeting)

Al terminar el sprint se tiene una junta llamada junta de revisión del sprint (Sprint Review meeting) la duración de esta junta depende de la duración del sprint que es 4 horas por cada 30 días y 1 hora por cada semana, en esta junta se realiza un demo del trabajo concluido (sprint backlog) y el PO verifica el incremento y que coincida con lo acordado en el criterio de aceptación y puede aceptarlo o rechazarlo y volverlo a agregar al backlog, algunos stakeholders relevantes podrían participar.

Junta de retrospectiva

También al terminar se tiene la junta llamada de retrospectiva del sprint la duración es la misma que la de revisión, donde el SM evalúa los procesos llevados durante el sprint, herramientas, colaboración, comunicación y algunas otros temas importantes, se discute que estuvo bien y que estuvo mal, aprender y hacer mejoras (lessons learned) para aplicar en los siguientes sprints, se documenta en el Scrum Guidance Body Document.

FASE 5. LANZAMIENTO.

Liberar al cliente los entregables

Una vez concluidos todos los sprits y teniendo todos los incrementos concluidos, podemos liberar al cliente nuestros entregables aceptados usando el método de deployment especificado por el cliente el proceso genera un documento llamado "Acuerdo de Entregable Trabajando" (Working Deliverables Agreement),

junta de evaluación retrospectiva del Sprint

Al final de cada sprint se tiene una junta de retrospectiva del Sprint donde Se evalúa los procesos llevados durante el Sprint, herramientas, colaboracion, comunicación y algunos otros temas importantes, se discute que estuvo bien y que estuvo mal que se puede aprender y hacer mejoras para aplicar en los siguientes Sprints.

Junta de Retrospectiva del Proyecto

Al final del proyecto se tiene una Junta de Retrospectiva del Proyecto donde Se evalúa los procesos llevados durante el proyecto, herramientas, colaboración, comunicación y algunas otros temas importantes, se discute que estuvo bien y que estuvo mal, que se puede aprender y hacer mejoras (lessons learned) para aplicar en los siguientes proyectos, se documentan las mejoras acordadas de acción" (Agreed Actionable Improvements).

MÉTODOS ESTIMACIÓN

1. Planning Pocket (Estimation Pocker)
2. fist of five (se usa para estimar y priorizar)
3. Wiband Delphi
4. Affinity Estimation

Capítulo 4

Métodos de Priorización

1. Esquema Simple
2. Esquema de priorizacion MoSCoW
3. Comparación por pares
4. Metodo de los 100 puntos.
5. Monopoly Money

Los 4 principales roles de Scrum

Para que este método funcione se han establecido una serie de roles que son los que garantizan que el método se cumple. Hay 3 roles principales y uno auxiliar. Son los siguientes:

Product owner

> Es el responsable del proyecto, administra, controla y comunica la Backlog List. Es el responsable de encontrar la visión del producto y reflejarla en la Backlog List.

Se trata de la persona que está en contacto directo con el cliente. Por tanto, tiene una tarea importante como interlocutor con todos los stakeholders del proyecto. Es quien conoce las peticiones y requerimientos de los clientes. Pero su labor más importante es maximizar el valor del trabajo del equipo de desarrollo.

Sólo debe haber un product owner para que los sprints funcionen. Tiene la responsabilidad de mantener el Product Backlog bien estructurado, detallado y priorizado.

Scrum Master

> Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que todo funciona según lo planeado.

Es el responsable de que la metodología Scrum sea comprendida y aplicada en la empresa. Es el mánager de Scrum. Por eso, su principal labor es ayudar en la adopción de esta metodología en todos los equipos. Para ello, no sólo formará al

equipo, sino que, también, debe ser el facilitador en todas las reuniones. Sus tareas básicas son:

- Gestionar el proceso Scrum para que aporte valor a la organización que lo adopta.
- Eliminar impedimentos.

Equipo de desarrollo

> Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de un Sprint.

Se trata de las personas encargadas de realizar las tareas. Debe ser un equipo multifuncional y autoorganizado y tienen la responsabilidad compartida de haber realizado el trabajo o no haberlo conseguido. De ahí que no se debe intervenir en sus dinámicas de funcionamiento. Es el equipo el que decide gestionarse internamente de una manera concreta. Y, por eso, tendrá que rendir cuentas como uno solo.

Stakeholders

Existen unos roles auxiliares que son aquellos que no tienen un rol formal y no se involucran en el proceso. Pero que, sin embargo, su opinión debe ser tomada en cuenta. Pueden ser desde expertos en negocio que pueden asesorar hasta clientes o proveedores. Algunos de ellos participan durante la revisión del sprint.

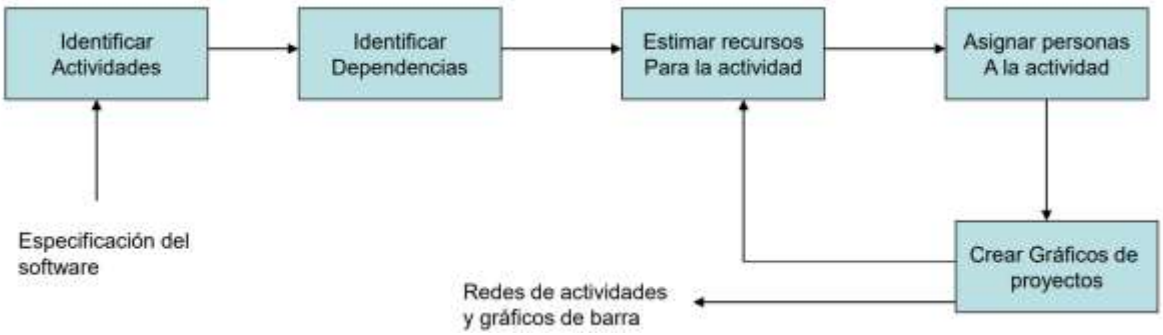
CALENDARIZACIÓN DE PROYECTOS DE SOFTWARE

El reciente aumento de aplicaciones en donde se utiliza la computadora ha sido posible debido a un hardware de bajo costo, por lo cual la demanda de software ha crecido de forma exponencial.

¿QUE ES LA CALENDARIZACIÓN?

> Es una actividad que distribuye estimaciones de esfuerzo a través de la duración planificada del proyecto, al asignar el esfuerzo a tareas específicas de ingeniería del software.

La calendarización implica separar todo el trabajo de un proyecto en actividades complementarias y considerar el tiempo requerido para completar dichas actividades (algunas se realizarán en paralelo), como se muestra en la siguiente gráfica.



CONSIDERACIONES

- Estimar la complejidad del problema y los costos de desarrollo de la solución.
- La productividad no es proporcional al número de personas que trabaja en una tarea.
- Agregar personas a un proyecto atrasado no soluciona los problemas.
- Lo inesperado siempre ocurre. Siempre debe considerarse las contingencias en la planificación.
- La calendarización se revisa cuando se refinan estimaciones, cuando cambian restricciones o cuando se reasignan recursos.

CRONOGRAMAS

Diagrama de Gantt

Un diagrama de Gantt se utiliza para representar visualmente los cronogramas del proyecto en una línea de tiempo. Un diagrama de Gantt en línea típico se divide en dos mitades. Las tareas se enumeran en el lado izquierdo en forma de hoja de cálculo tradicional, pero la línea de tiempo a la derecha ofrece una manera rápida y fácil de entender el cronograma del proyecto en su totalidad.

Tarea	Predec.	Duración
A	-	2
B	A	3
C	-	2
D	C	3
E	D_{i+1}	2
F	B_{F1-1}	3
G	D, E, F	3
H	G_{FF}	2

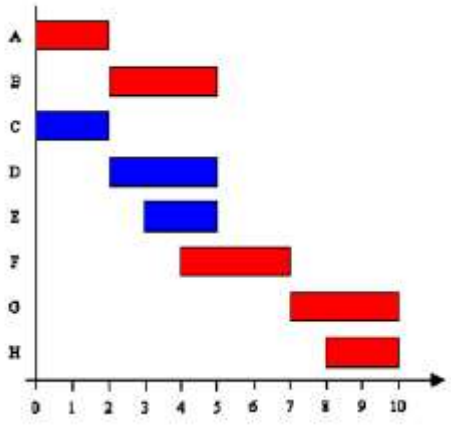
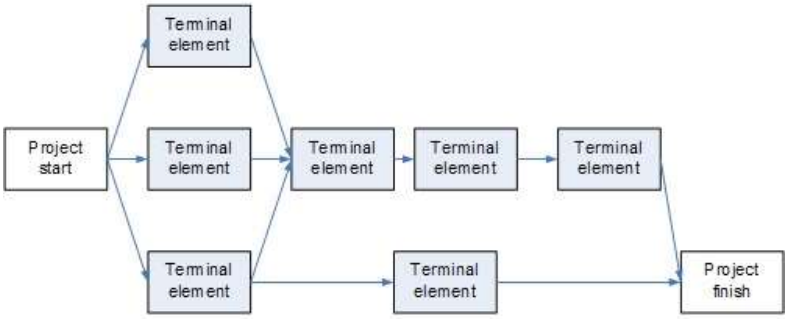


Gráfico PERT

PERT significa Técnica de revisión de evaluación de programas y se refiere al método de utilizar un gráfico para trazar las tareas que deben realizarse para completar un proyecto. Utiliza flechas para mostrar las tareas necesarias para llegar a un evento, que están simbolizadas por nodos. Un nodo representa una fase de proyecto completada. Se utilizan para ayudar a estimar el tiempo que se tarda en completar las tareas de un proyecto.

Diagrama de red

Un diagrama de red es un esquema que muestra todas las tareas de un proyecto, quién es responsable de ellas y el flujo de trabajo necesario para completarlas. En otras palabras, ayudan a visualizar el cronograma del proyecto. Al igual que el gráfico PERT, también se compone de flechas y nodos que muestran el curso de las tareas a lo largo del ciclo de vida de un proyecto. Se puede usar para seguir el progreso y el alcance una vez que se ha ejecutado un proyecto.



En el desarrollo de ingeniería de software de un producto, puede utilizar la que mas se adapte a sus necesidades de acuerdo con la metodología elegida.

BIBLIOGRAFÍA

Beck, K. (2000). *Extreme programming eXplained : embrace change*. Reading, MA : Addison-Wesley. https://archive.org/details/extremeprogrammi00beck/mode/2up

López Sanz, M. (2000). *Ciclo de vida del software*. *1*.

Proyectosagiles. (2021). *Historia de Scrum*. https://proyectosagiles.org/historia-de-scrum/

Rcasalla. (2020). *Desarrollo de Software en Equipos*. https://rcasalla.gitbooks.io/libro-desarrollo-de-software/

Sommerville, I. (2005). *Ingeniaria del Software* (7 Edición). https://archive.org/details/pdfy-g7E-3nrc3Q4yRJ4y/page/n4/mode/1up