MÓDULO 1. FUNDAMENTOS DE LA INGENIERÍA DE SOFTWARE

Imagen tomada de Freepik

DESCRIPCIÓN DEL MÓDULO 1

La ingeniería de software es una disciplina de la ingeniería cuyo objetivo es el desarrollo rentable de sistemas de software. El software se considera abstracto e intangible, así mismo, no está limitado por materiales, gobernado por leyes físicas o por procesos de fabricación. En ocasiones esta falta de restricciones naturales significa que el software puede volverse complejo y, por lo tanto, difícil de entender. Aun así, es una disciplina con la que nos vemos beneficiados por las múltiples aplicaciones que nos encontramos rodeado.

En este módulo se proporciona los conceptos relacionados a la ingeniería de software. También las actividades relacionadas el desarrollo del software, el ciclo de vida y metodologías de desarrollo del software.

Contenido

Unidad 1. Definición de software

La ingeniería de software es una disciplina de la ingeniería cuyo objetivo es el desarrollo rentable de sistemas de software. El software se considera abstracto e intangible. Es una disciplina con la que vivimos por las múltiples aplicaciones que nos rodean.

La ingeniería de software es una disciplina que surgió a fines de la década de 1960 como una nueva disciplina de ingeniería que se ocupa de todos los aspectos relacionados con la producción de software. Abarca conceptos, principios, teorías, técnicas y herramientas que se pueden utilizar para desarrollar software profesional de alta calidad.

El concepto de esta disciplina fue presentado por primera vez en la Conferencia de ingeniería de software de la OTAN de 1968 en Garmisch, Alemania. La ingeniería de software hace hincapié en *un enfoque sistemático y disciplinado para el desarrollo y la evolución del software* y, por lo general, se aplica a la *construcción de grandes sistemas (o productos) de software en los que participan* equipos de numerosos ingenieros de software.

1.1. Definición de Ingeniería de software

El concepto de ingeniería de software se desarrolló para abordar problemas asociados con proyectos de software de baja calidad. Cuando un software funciona demasiado rápido, gasta demasiado o tiene mala calidad, hay un problema. En este sentido, la ingeniería de software es una disciplina que integra la ingeniería para la creación de productos intangibles tipo software, cuyo objetivo se centra en crear soluciones para disminuir las problemáticas asociadas al desarrollo de software que satisfagan las necesidades de los clientes.

Según señala la IEEE, la lingeniería de software es: 1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software. 2) El estudio de enfoques según el punto 1 [1].

Unidad 2. Desafíos de la ingeniería de software

Como indica Sommerville en [22], la ingeniería de software en el siglo XXI enfrenta tres desafíos importantes que se mencionan a continuación.

1. El desafío de lo heredado: muchos de los sistemas de software que se utilizan hoy en día se desarrollaron hace años, sin embargo, realizan funciones comerciales críticas. Por lo que el desafío de lo heredado es de mantener y actualizar este software de tal manera que se eviten los costos excesivos y se continúen brindando los servicios comerciales esenciales.

2. El desafío de la heterogeneidad: cada vez más, se requiere que los sistemas operen como sistemas distribuidos a través de redes que incluyen diferentes tipos de computadoras y con diferentes tipos de sistemas de soporte. El desafío de la heterogeneidad es el desafío de desarrollar técnicas para crear software confiable que sea lo suficientemente flexible para hacer frente a esta heterogeneidad.

3. El desafío de la entrega: muchas técnicas tradicionales de ingeniería de software consumen mucho tiempo. El tiempo que toman es necesario para lograr la calidad del software. Sin embargo, las empresas de hoy deben ser receptivas y cambiar muy rápidamente, por lo que su software de soporte debe cambiar con la misma rapidez. El desafío de la entrega es el desafío de acortar los tiempos de entrega para sistemas grandes y complejos sin comprometer la calidad del sistema.

Estos desafíos no son independientes. Por ejemplo, puede ser necesario realizar cambios rápidos en un sistema heredado para que sea accesible a través de una red. Para hacer frente a estos desafíos, necesitaremos nuevas herramientas y técnicas, así como formas innovadoras de combinar y utilizar los métodos de ingeniería de software existentes.

Unidad 3. Conceptos de la ingeniería de software

1.3. Conceptos de Ingeniería de Software

El desarrollo de software requiere la colaboración de muchas personas con diferentes intereses y antecedentes. Se incluyen los clientes que solicitan y pagan por el sistema. A cada participante se le asigna un conjunto de responsabilidades (rol) a contemplar durante el proceso.

1.3.1. Participantes y papeles

Las asignaciones del personal a las tareas en un proyecto de software dependen de la dimensión del proyecto. Tener en cuenta el concepto de participantes y papeles se menciona a continuación.

Participantes	Qué incluye	Ejemplo
Todos aquellos que están involucrados en el proyecto software	- Cualquier persona o entidad involucrada - Tiene un rol	- Ícaro Medeiro - Rosa Lara - Compañía de trenes - Viajeros

Papeles o roles	Qué incluye	Ejemplo
Define un conjunto de responsabilidades, es decir tareas pendientes	- Conjunto de responsabilidades en el proyecto o sistema - Asociado a un conjunto de tareas - Asigna una tarea a un participante - Un participante puede asumir varios roles	Rol: "Tester" Responsable: - Escribir pruebas - Reportar fallas - Comprobar si las correcciones de errores abordan una falla específica Rol: Arquitecto del sistema Responsable: - Garantizar la coherencia en las decisiones de diseño y definir - interfaces de subsistema - Formular la estrategia de integración del sistema

Una vez se ha decidido acerca de los papeles o roles de los miembros del proyecto de software, se debe decidir que tipo de personal se necesita en cada papel o rol. Para ello se presentan diferentes participantes de acuerdo con las etapas de su desarrollo. En el siguiente cuadro se presentan algunos roles a contemplar en el desarrollo de un software.

Algunos roles de personas en un software

- **Cliente**: *desea y solicita la construcción del software*
- **Líder del proyecto**: crear un plan, coordina el equipo
- **Analistas de requisitos**: i*nteractuar con el cliente y así conocer necesidades del sistema a desarrollar*
- **Diseñadores**: define estrategias de cómo sería el funcionamiento del software.
- **Desarrollador**: diseña y desarrolla *el código o partes de acuerdo con los requisitos*
- **Tester**: *verifica aspectos para aseguramiento de calidad*
- **Usuario:** *compra y usa el producto software*

1.3.2. Sistemas y modelos

En este punto se presentan los conceptos sistemas y modelos.

Un **sistema** se refiere a un conjunto de entidades de interés que interactúan y conforman un todo interés para alcanzar un objetivo. Los sistemas pueden interactuar con otros sistemas o ser componentes de sistemas más grandes.

- Ejemplo de un sistema: Distribuidor de entradas del tren, sistema de ATM del banco.

Sistema es un grupo de objetos o componentes que, juntos, forman un todo. Están formados por componentes y tienen límites definidos. [2]. En los sistemas informáticos, estos elementos incluyen hardware, software, personas, instalaciones y procesos.

Para comprender los modelos, es importante definir el sistema y comprender las limitaciones del modelo para representar el sistema.

Los **modelos** muestran los procesos en general que son soportados por el sistema. Hace referencia a cualquier abstracción de la realidad compuesto de planos o vistas, diagramas y

esquemas, gráficas e imágenes. Algunos de los tipos de modelos que se utilizan en ciencia e ingeniería incluyen modelos conceptuales como dibujos y diagramas.

- Ejemplo de modelo:
  - planos para el distribuidor de boletos de tren
  - esquemas para alambrado eléctrico
  - modelos de objetos de software
  - diagramas de caso de uso
  - modelo de diseño de la Base de Datos

Modelo de sistema representa aspectos de un sistema y su entorno. Hay diferentes tipos de modelos, ya que hay una variedad de propósitos para los que se construyen. Es útil tener una forma común de hablar sobre modelos ya que permiten la comprensión del comportamiento del sistema. [3]

1.3.3. Productos de trabajo

Los productos de trabajo son **resultados** relacionados con el software. Es un artefacto (concreto) que se genera durante el desarrollo. Sobre los productos:

- Puede ser un documento o fragmento de software para otros desarrolladores o para el cliente según el caso.
- Se genera para consumo o uso interno del proyecto como producto de trabajo interno
- Producto de trabajo para un cliente = entregable (delivery)

Ejemplos de productos trabajo de sus siglas en inglés Work Package:

- Manuales de operación y mantenimiento
- Distribuidores de boletos y su software
- Reporte

1.3.4. Actividades, tareas y recursos

Hay muchas formas de trabajar las actividades de un proyecto. Estas actividades son la unidad principal de trabajo. Implica dividir en tareas (suelen ser una pieza mucho más pequeña de trabajo) y la utilización de recurso para completar una actividad, lo que se representa y ejemplifica en las imágenes siguientes.

Actividad	⟺	Tarea	⟺	Recurso
- Conjunto de tareas. - Se realiza con un propósito		Representa una unidad simple de trabajo que puede ser administrada		Bienes que se usan para realizar un trabajo o tarea Ejemplo: bienes, insumos

Ejemplo de: actividad, tarea y recurso

Actividad	⟺	Tarea	⟺	Recurso
Obtención de requerimientos. - Propósito: *definir con el cliente lo que necesita el software*		Entrevista con el cliente. Asignación que hace un gerente a un analista		Tiempo, Grabadora

Se dice entonces:

- Cada Actividad a su vez se compone de una serie de tareas.
- Que una tarea consume recursos y produce un producto de trabajo
- Los recursos son participantes, tiempo, equipo

1.3.5. Objetivos, requerimientos y restricciones

En los siguientes puntos se describen los aspectos acerca de objetivos, requerimientos y restricciones.

Objetivos.

- Se usa para guiar el proyecto
- Definen los atributos del sistema que son importantes

En ocasiones parte de los problemas la complejidad en el desarrollo del software viene de objetivos mal definidos

Requerimientos o requisitos

Se refiere a características que debe tener el sistema

Los requisitos **especifican** qué construir:

- Decir "qué" y no "cómo"
- Decir el problema, no la solución
- Reflejan el diseño del sistema, no el diseño del software

Requisitos ayudan

- Comprender con precisión lo que se requiere el software
- Comunicar este entendimiento a las partes del desarrollo
- Controlar la producción para asegurar que el sistema cumple con las especificaciones definidas

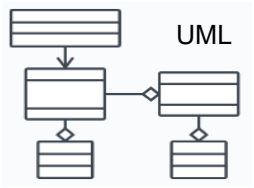
Restricciones

Se refiere a principios a considerar para el desarrollo de la solución

- Limitación a considerar para el desarrollo y que debe poseer el sistema.
- Puede ser, además, una forma de control.

Ejemplo: restricción para reducir el consumo de recurso

1.3.6. Notaciones, métodos y metodologías



Notaciones. Son un conjunto de reglas gráficas o textuales para *representar un modelo*

Ejemplo: - Alfabeto romano para representar palabras

- UML para representar modelos Orientados a Objetos

Método. Se refiere a una técnica repetible para representar un problema específico

Ejemplo: - receta: método cocinar plato particular

- Algoritmo ordenamiento: método para ordenar elementos de una lista

Metodología. Es una colección de métodos para solución de una clase de problemas

Ejemplo: - Metodología Booch, proceso de desarrollo de software unificado

- Libro de recetas de mariscos: metodología para preparación de marisco
- Técnica de Modelado de objetos (OMT)

Metodologías de desarrollo de software: descomponen el proceso en actividades para el desarrollo del software

Técnica de modelado de objetos (OMT) proporciona métodos para tres actividades:

- Análisis: enfoca formalización de requerimientos del sistema en un modelo de objeto
- Diseño de sistema: decisiones estratégicas
- Diseño del objeto: transforma el modelo de análisis en un modelo objeto que puede ser puesto en práctica

1.3.7. Concepto de ciclo de vida del software

Se refiere al proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema. El ciclo de vida define una metodología para mejorar la calidad de software y el proceso de desarrollo en general.

> El ciclo de vida del software consiste en un plan detallado que describe cómo desarrollar, mantener, reemplazar y alterar o mejorar software específico. [4]

Unidad 4. Actividades de desarrollo de la ingeniería de software

Para el desarrollo de software se necesita realizar una serie de actividades, como en todo proyecto, que conlleven la ejecución desde su inicio hasta su implementación. Debido a que no es un proceso automático, se desarrollan una serie de actividades que se mencionan en los siguientes puntos.

1.4. Actividades de desarrollo de la Ingeniería de Software

Algunas de las actividades en el proceso de desarrollo de software son de naturaleza secuencial, dividiendo el proceso en fases como en la figura. Estas fases se mencionan en los puntos siguientes de acuerdo como describe [11]. (agregar ref del libro verde)

Figura: Fases en el desarrollo de software

1.4.1. Obtención de requerimientos

Actividad en la que el cliente y desarrolladores definen el propósito del sistema. También se definen requerimientos no funcionales

Objetivo: captar las necesidades y deseos del cliente

Resultado de esta actividad: descripción del sistema en término de actores y casos de uso.

1.4.2. Análisis de los requerimientos

Se crea la especificación de las capacidades funcionales necesarias del software.

Objetivo: Producir un modelo de sistema correcto, completo, realista, sin ambigüedades y confiable.

Resultado: Transformar los casos de uso dentro de un modelo de objeto que describa completamente el sistema.

1.4.3. Diseño del software

Se definen objetivos de diseño donde se descompone el sistema en subsistemas que pueden ser realizados por equipos pequeños.

Objetivo: Seleccionar estrategias para la construcción del sistema.

Resultados: -diagrama de organización que representa la relación entre el hardware y software del sistema.

1.4.4. Desarrollo del software

Etapa en la que se determina características internas como algoritmos y método para el desarrollo del software.

Objetivo: definir la estructura interna y los algoritmos de los componentes que cumplen con las especificaciones orientadas al cliente.

Producto: especificaciones orientadas a la implementación para componentes.

1.4.5. Pruebas y aseguramiento de la calidad del software

Se verifica los requisitos de realismo, consistencia e integridad.

Objetivo: descubrir errores en el documento de requisitos.

Producto: Cuando se encuentran errores, se debe modificar para corregir estos problemas

1.4.6. Despliegue o puesta en marcha

Se refiere a la implementación. Incluye la implementación de los atributos y métodos de cada objeto, así como la integración de todos los objetos de forma tal que funcionen como un solo sistema.

Objetivo: producir componentes codificados que implementen con precisión el diseño orientado a la implementación

Producto: Traducción del modelo de objetos en código fuente

1.4.7. Mantenimiento de software

Implica corregir errores que no se descubrieron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema y mejorar los servicios del sistema a medida que se descubren nuevos requisitos.

Objetivo: solucionar problemas y mantener los sistemas actualizados

Producto: producto debe mantenerse operativo y se requiere mantenimiento del producto

A continuación, se describen tipos de mantenimiento del software según se describe en [5] y [6]

1.4.7.1. Mantenimiento correctivo

El mantenimiento correctivo del software es lo que normalmente se asociaría con el mantenimiento de cualquier tipo. Los cambios correctivos en el mantenimiento del software son aquellos que corrigen errores, fallas y defectos en el software. A menudo se presenta en forma de actualizaciones pequeñas y rápidas de forma semirregular [5].

Este tipo de mantenimiento [6].

- Aborda los errores y fallas dentro de las aplicaciones de software que podrían afectar varias partes de su software, incluido el diseño, la lógica y el código.
- Estas correcciones generalmente provienen de informes de errores creados por usuarios o clientes, pero el mantenimiento correctivo del software puede ayudar a detectarlos antes que sus clientes, lo que puede ayudar a la reputación de su marca.

1.4.7.2. Mantenimiento preventivo

Se refiere a los cambios de software realizados para preparar su producto para el futuro. Por lo tanto, los cambios de mantenimiento de software son preventivos cuando se preparan para cualquier cambio potencial que se avecina. Esto incluye hacer que su código sea más fácil de escalar o mantener y administrar su contenido heredado. También cubre la búsqueda y reparación de fallas latentes en su producto, antes de que se conviertan en fallas operativas. [5]

1.4.7.3. Mantenimiento adaptativo

Los cambios adaptativos se centran en la infraestructura del software. Se crean en respuesta a nuevos sistemas operativos, nuevo hardware y nuevas plataformas, para mantener la compatibilidad del programa [5].

El mantenimiento de software adaptativo se vuelve importante cuando cambia el entorno de su software. Esto puede ser provocado por cambios en el sistema operativo, hardware, dependencias de software, almacenamiento en la nube o incluso cambios dentro del sistema operativo. A veces, el mantenimiento de software adaptativo también refleja las políticas o reglas de la organización. La actualización de servicios, la modificación de proveedores o el cambio de procesadores de pago pueden requerir un mantenimiento de software adaptable [6].

1.4.7.4. Mantenimiento perfectivo

Se aborda la funcionalidad y la usabilidad del software. Implica cambiar la funcionalidad del producto existente refinando, eliminando o agregando nuevas funciones [5].

Se enfoca en la evolución de los requisitos y características que existen en su sistema. A medida que los usuarios interactúan con sus aplicaciones, pueden notar cosas que no se notó o sugerir nuevas funciones que les gustarían como parte del software, que podrían convertirse en futuros proyectos o mejoras. [6]

Unidad 5. Modelos de ciclos de vida de desarrollo de software

En el desarrollo de software, existen diferentes sistemas utilizados en el proceso de desarrollo de software, conocidos como Modelos de Desarrollo de Software que se mencionan a continuación.

Tener en cuenta que un **Modelo de proceso** es una forma estandarizada para el planeamiento, organización, así como la ejecución del desarrollo de un proyecto de software

Un **modelo de proceso** incluye el tiempo de vida entera de un producto, desde su concepción hasta la instalación, por lo que estos procesos también son conocidos como: **ciclo de vida** de un producto, en este caso, un producto tipo software.

Ciclo de vida ≈ modelo de proceso

1.5. Modelos de ciclos de vida de desarrollo de software

Estos modelos según la particularidad que los diferencia, siguen un paso definitivo para garantizar la finalización del desarrollo de software.

1.5.1. Clasificación de los ciclos de vida del software

Algunos modelos referentes con el ciclo de vida de software son: el modelo en cascada, incremental, iterativo, espiral, según su clasificación.

1.5.1.1. Modelo en Cascada

El modelo de cascada fue el primer modelo de proceso que se introdujo. También se conoce como un modelo de ciclo de vida secuencial lineal. Es muy simple de entender y usar. En un modelo en cascada, cada fase debe ser completado antes de que pueda comenzar la siguiente fase y no hay superposición en las fases [4].

Resultado: el resultado de una fase actúa como entrada para la siguiente fase secuencialmente.

Cascada (Waterfall)

Proceso en un flujo secuencial lineal. Esto significa que cualquier fase del proceso de desarrollo comienza solo si se completa la fase anterior. En este modelo de cascada, las fases no se superponen.

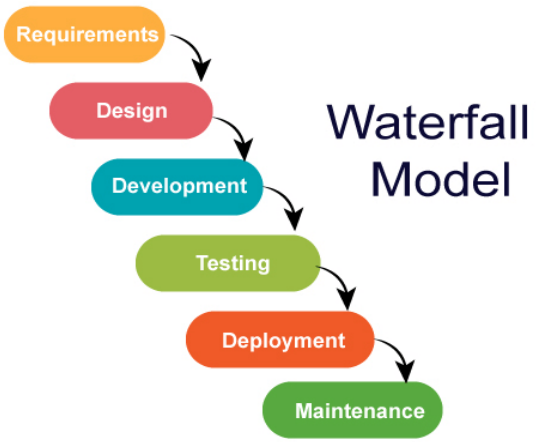


Figura. Fases y comportamiento de modelo en cascada. Imagen adaptada de [7]

1.5.1.2. Modelo Incremental

Es una evolución del modelo en cascada. Surgió como una manera de reducir la repetición del trabajo en el proceso de desarrollo y no se necesita esperar por todos los módulos para ser desarrollados e integrados [8].

Como parte del modelo Incremental, cada módulo (incremento) pasa por varias fases: requisitos, diseño y desarrollo, pruebas e implementación. Cada nueva versión del módulo agrega

funcionalidad al módulo de versión anterior. El proceso continúa hasta que se implementa toda la funcionalidad prevista y se desarrolla el sistema completo [9].

Incremental

Cada módulo es una sección autónoma por lo que no hay dependencia en otros módulos, por lo tanto, se puede entregar el proyecto con un desarrollo inicial y otros desarrollos pueden ser agregados en otras entregas.

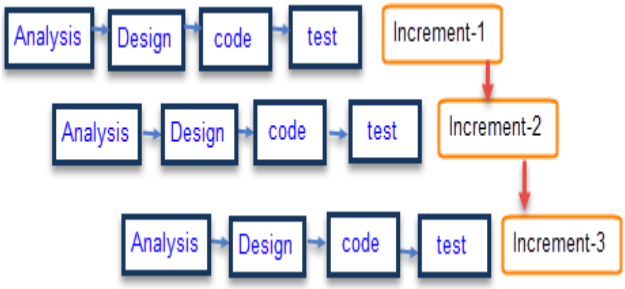


Figura. Fases del modelo incremental. Imagen adaptada de [10]

1.5.1.3. Modelo Iterativo

En el modelo iterativo, el proceso iterativo comienza con una implementación simple de un pequeño conjunto de requisitos de software y mejora iterativamente las versiones en evolución hasta que se implementa el sistema completo y está listo para implementarse [11]. Permite acceder a fases anteriores, en las que se realizan las variaciones respectivamente. Luego, el prototipo generado se revisa más a fondo para conocer los requisitos adicionales. La práctica luego toma una forma iterativa para crear una nueva versión de la aplicación.

El producto final que se crea de forma iterativa cumple con los requisitos del usuario. El resultado final es el proyecto renovado al final del proceso de ciclo de vida de desarrollo de software.

Iterativo

Comienza con requisitos mínimos que especifican e implementan solo una parte del software. Posteriormente se continúa con el resto de los requisitos hasta completar el proyecto. Útil ya que puede aceptar modificaciones en los requisitos originales del sistema.

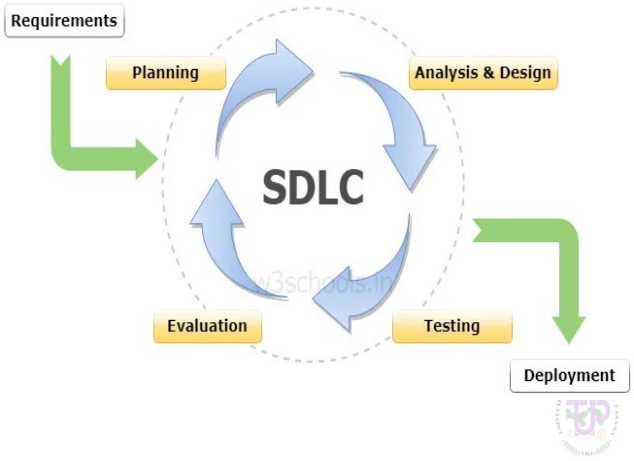


Figura. Modelo iterativo. Figura tomada de [12]

1.5.1.4. Modelo Espiral

El modelo en espiral, propuesto inicialmente por Boehm, es un modelo de proceso de software evolutivo que combina la característica iterativa de la creación de prototipos con los aspectos controlados y sistemáticos del modelo secuencial lineal (cascada). Implementa el potencial para el desarrollo rápido de nuevas versiones del software. [13] El proyecto se entrega en bucles, donde cada bucle son las fases del proceso de desarrollo de software.

Con el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión adicional puede ser un modelo o prototipo en papel. Con iteraciones posteriores, se producen versiones cada vez más completas del. Cada ciclo espiral en la iteración es una etapa en el proceso de desarrollo de software que tiene cuatro cuadrantes, y cada uno de ellos representa alguna etapa específica del desarrollo de software.

Espiral

Permite desplegar y refinar el producto en cada fase de la espiral, con la capacidad de construir prototipos en cada etapa. Se crea un prototipo al inicio de cada fase como técnica de gestión de riesgos.

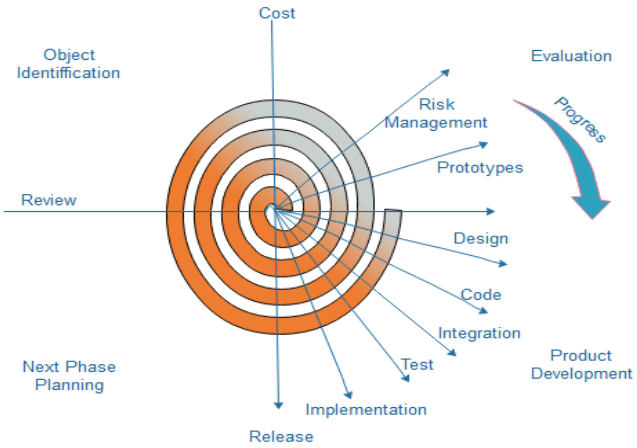


Figura. Modelo espira, tomado de [13]

Actividades comunes entre diferentes modelos de procesos de desarrollo de software

(1)- *Especificación del software*: definir la funcionalidad del software y restricciones en sus operaciones
(2)- *Diseño e implementación*: producir software que cumpla especificaciones
(3)- *Validación del software*: software debe validarse para asegurarse qué es lo que el cliente quiere
(4)- Evolución del software: Evolucionar para cumplir cambios requeridos por el cliente

Unidad 6. Metodologías de desarrollo de software

1.6. Metodología de desarrollo de software

Una **metodología** es un sistema de métodos y principios utilizados en una subdisciplina particular del diseño de software. Las metodologías en el entorno de trabajo profesional actual son medios para guiar a las personas, los recursos y los procesos de manera definida y repetible. [14]

1.6.1. Concepto de metodología de desarrollo de software

Consiste en la estructura de actividades de desarrollo completo. Una metodología de desarrollo de software permite la planificación, el análisis, el diseño, el desarrollo, la implementación, las pruebas y el mantenimiento del proceso de desarrollo de software aportando reglas y directrices adecuadas. Estas metodologías ayudan a gestionar un proyecto de software [15].

Una metodología de desarrollo de software puede definirse como una guía adecuada para mantener un proceso de desarrollo sucesivo. Es un conjunto de tareas organizadas comprometidas con funciones del proceso de desarrollo de software

1.6.2. Metodologías tradicionales

1.6.2.1. Proceso Unificado (Rational Unified Process – RUP)

El proceso unificado es un estilo tradicional de diseño incremental impulsado por la construcción de vistas de una arquitectura de sistema. Su enfoque iterativo e incremental permite una comprensión cada vez mayor del problema a través de refinamientos sucesivos.

1.6.2.1.1. Concepto

El Rational Unified Process (RUP) es un proceso de ingeniería de software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es garantizar la producción de software de alta calidad que satisfaga las necesidades de sus usuarios finales, dentro de un calendario y presupuesto predecibles [18].

1.6.2.1.2. Fases y disciplinas

Las fases de RUP se describen a continuación según menciona [19]

- La fase de **inicio** establece la lógica comercial del proyecto, delimita el alcance del proyecto y un prototipo conceptual
- La fase de **elaboración** recopila requisitos más detallados, realiza un análisis y diseño de alto nivel para establecer una línea base de arquitectura y crear un plan para la construcción.
- La fase de **construcción** consta de muchas iteraciones, en las que cada iteración analiza, diseña, construye, prueba e integra un subconjunto de requisitos de un proyecto
- La fase de **transición** incluye pruebas beta, empaquetado, ajuste del rendimiento y capacitación del usuario

1.6.3. Metodologías ágiles

El término ágil surge como iniciativa de un conjunto de expertos en el área de desarrollo de software con el fin de optimizar el proceso de creación del mismo, el cual era caracterizado por ser rígido y con mucha documentación.

1.6.3.1. XP (Extreme Programming)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo [16].

XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

XP se **define** como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

1.6.3.2. FDD (Feature Driven Development)

Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software [17].

1.6.3.3. Scrum

Está especialmente indicada para proyectos con un rápido cambio de requisitos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración [16].

1.6.3.4. Crystal Methodology

Crystal es una metodología en la cual se establecen códigos de color como parte de la definición de la complejidad de la misma, si es más oscuro entonces el método es más pesado. Cuánto más crítico es el sistema más rigor se necesita. Además, cristal sugiere que se defina un color para cada proyecto en función de su criticidad y tamaño [17].

Crystal reúne metodologías ágiles de desarrollo de software que son posibles de usar para diversos proyectos de software dependiendo del tamaño, complejidad, capacidad de gestión de

riesgos y número de personas involucradas. Consta de varios métodos para elegir la metodología adecuada para el proyecto seleccionado. Todos y cada uno de los miembros del equipo de la familia Crystal, seleccionan diferentes colores según la carga de trabajo asignada [15].

1.6.3.5. Dynamic Systems Development Method (DSDM)

Esta metodología Nace en 1994 con el objetivo de crear una metodología RAD unificada. Es un marco que incorpora gran parte del conocimiento sobre la gestión de proyectos. Se basa en las mejores prácticas para comenzar a implementar una estructura de proyecto, siendo sus puntos fuertes la simplicidad, la extensibilidad, probado en el pasado, pero sin pretender ser la solución a todo tipo de proyectos [14].

Sus principales características son:

- Es un proceso iterativo e incremental donde el equipo de desarrollo y el usuario trabajan juntos.
- Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases

DSDM se presenta como el método de desarrollo ágil más maduro, mientras que muchas metodologías ágiles son más bien metodologías de programación que modelos de procesos.

1.6.3.6. Lean Software Development

Surge a partir de la experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. Los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios [16].

Unidad 7. Lenguaje de Modelado Unificado (UML)

1.7. Lenguaje de Modelado Unificado (UML)

El lenguaje de modelado unificado (UML) es un lenguaje de modelado de propósito general en el campo de la ingeniería de software. Es un estándar abierto que incluye un conjunto de técnicas de notaciones gráficas para crear modelos visuales de sistemas de software orientados a objetos.

UML tomó las notaciones de la técnica de modelado de objetos, la ingeniería de software orientada a objetos y el método de Booch.

Breve historia

UML fue inventado por James Rumbaugh, Grady Booch e Ivar Jacobson. Después de que Rational Software Corporation contratara a James Rumbaugh de General Electric en 1994, la empresa se convirtió en la fuente de los dos enfoques de modelado orientado a objetos más populares del momento: la técnica de modelado de objetos (OMT) de Rumbaugh, que era mejor para el análisis orientado a objetos (OOA), y el método Booch de Grady Booch, que era mejor para el diseño orientado a objetos (OOD). Pronto fueron asistidos en sus esfuerzos por Ivar Jacobson, el creador del método de ingeniería de software orientado a objetos (OOSE). Finalmente, Jacobson se incorporó a Rational en 1995, después de que su compañía Objectory AB, fuera adquirida por Rational [21].

UM, también, ofrece una forma estándar de visualizar los planos arquitectónicos de un sistema, incluidos elementos como actividades, actores, procesos comerciales, esquemas de bases de datos, componentes, declaraciones de lenguaje de programación y componentes de software reutilizables.

> UML se utiliza para especificar, visualizar, modificar, construir y documentar los artefactos de un sistema de software orientado a objetos en desarrollo [21].

Tipos de diagramas

Hay dos tipos de diagramas utilizados en UML: diagramas de estructura y diagramas de comportamiento.

- Los Diagramas de Estructura: representan los elementos que componen el sistema.
- Los diagramas de comportamiento: representan los procesos que se desarrollan en un entorno modelado. El grupo Diagramas de Comportamiento incluye:

Diagramas de estructura	*Diagramas de comportamiento*
• Diagrama de clase • Diagrama de componentes • Diagrama de implementación • Diagrama de objetos • Diagrama de paquete	• Diagrama de actividad • Diagrama de máquina de estado • Use el diagrama de caso de uso • Diagrama general de interacción .

El subgrupo de diagramas de interacción controla el flujo de control y datos e incluye:

- Diagrama de comunicación
- Diagrama general de interacción
- Diagrama de secuencia
- Diagrama de tiempo

Conclusiones

En este módulo se abordaron temas referentes al concepto de ingeniería de software hasta lo relacionado al lenguaje de proceso unificado UML. La ingeniería de software es una actividad de ingeniería que sigue las etapas básicas para el desarrollo de un problema como en un proyecto.

Cualquier proceso de software incluye al menos cuatro tipos de actividades. Estas son la especificación de software, donde los clientes e ingenieros definen el software que se va a producir y las limitaciones de su funcionamiento. Luego el desarrollo de software, donde se diseña y programa el software. Seguido de la validación de software, donde se comprueba el software

para asegurarse de que es lo que requiere el cliente. Luego evolución del software, donde el software se modifica para reflejar los requisitos cambiantes del cliente y del mercado.

Posteriormente se abordan las metodologías para el desarrollo de software. Se muestran cómo van evolucionando hasta alcanzar el proyecto final de software. Estas metodologías se eligen de acuerdo con el tipo de proyecto que se desarrolla.

El lenguaje de modela UML es ampliamente utilizado por ingenieros de software, desarrolladores de software y diseñadores de software. Es decir, por aquellos especialistas que necesitan crear la documentación detallada del software, para especificar el ciclo de vida del desarrollo del software. Por lo que este lenguaje es de utilidad para apoyarse en la documentación del software que se va a desarrollar

Referencias

[1] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (1999). The guide to the Software Engineering Body of Knowledge. *IEEE software*, *16*(6), 35–44. https://doi.org/10.1109/52.805471 https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf

[2] System and system models. (n.d.). *Hawaii.edu*. Retrieved May 14, 2022, from https://manoa.hawaii.edu/exploringourfluidearth/standards-alignment/next-generation-science-standards-ngss/crosscutting-concepts/system-and-system-models

[3] System modeling concepts. (n.d.). *Sebokwiki.org*. Retrieved May 14, 2022, from https://www.sebokwiki.org/wiki/System_Modeling_Concepts

[4] *Kanchiuniv.ac.in*. Retrieved May 14, 2022, from https://kanchiuniv.ac.in/coursematerials/SOFTWARE%20ENGINEERING_course_material.pdf

[5] Reed, N. (2018, October 2). The 4 software maintenance categories and what they mean for your users. *Parker Software*. Retrieved May 15, 2022, from https://www.parkersoftware.com/blog/the-4-software-maintenance-categories-and-what-they-mean-for-your-users/

[6] The 4 types of software maintenance & how they help. (n.d.). *Default*. Retrieved May 15, 2022, from https://www.castsoftware.com/glossary/Four-Types-Of-Software-Maintenance-How-They-Help-Your-Organization-Preventive-Perfective-Adaptive-corrective

[7] Jayathilaka, C. (2020, July 30). Waterfall methodology - chathmini jayathilaka. *Medium*. Retrieved May 15, 2022, from https://medium.com/@chathmini96/waterfall-vs-agile-methodology-28001a9ca487

[8] What is Incremental Model in software testing and what are advantages and disadvantages of Incremental Model. (2015, February 7). *Testingfreak*. Retrieved May 15, 2022, from https://testingfreak.com/incremental-model-software-testing-advantages-disadvantages-incremental-model/

[9] Incremental Model in software engineering. (2022, February 24). InterviewBit. Retrieved May 15, 2022, from https://www.interviewbit.com/blog/incremental-model/

[10] Martin, M. (2020, January 31). Incremental model in SDLC: Use, advantage & disadvantage. *Guru99*. Retrieved May 15, 2022, from https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html

[11] Software Development Process. (n.d.). *Umn.edu*. Retrieved May 22, 2022, from

https://www.d.umn.edu/~gshute/softeng/process.html

[12] SDLC Iterative Model. (n.d.). *W3schools.in*. Retrieved May 15, 2022, from https://www.w3schools.in/sdlc/iterative-model

[13] Spiral model. (n.d.). *www.javatpoint.com*. Retrieved May 15, 2022, from https://www.javatpoint.com/software-engineering-spiral-model

[14] Voigt, B. J. J., Glinz, M., & Seybold, D.-I. C. (2004). Dynamic system development method. *Uzh.ch*. Retrieved May 20, 2022, from https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws03/14_Voigt_DSMD_Ausarbeitung.pdf

[15] Dilmini Rathnayaka, G. U., & Kumara, B. (n.d.). A review of software development methodologies in software engineering. Ijariie.com. Retrieved May 20, 2022, from http://ijariie.com/AdminUploadPdf/A_Review_of_Software_Development_Methodologies_in_Software_Engineering_ijariie12553.pdf

[16] (N.d.). Upv.es. Retrieved May 20, 2022, from https://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf

[17] (N.d.). *Org.sv*. Retrieved May 20, 2022, from http://www.redicces.org.sv/jspui/bitstream/10972/2917/1/Articulo1.pdf

[18] Anwar, A. (n.d.). A review of RUP (Rational Unified Process). Cscjournals.org. Retrieved May 20, 2022, from https://www.cscjournals.org/manuscript/Journals/IJSE/Volume5/Issue2/IJSE-142.pdf

[19] Madar, J. (n.d.). Rational unified process computing science 213. *Concordia.ca*. Retrieved May 20, 2022, from https://users.encs.concordia.ca/~paquet/wiki/images/3/31/RUP.pdf

[20] SDLC - Iterative Model. (n.d.). Tutorialspoint.com. Retrieved May 15, 2022, from https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

[21] CS302: Introduction to UML in software engineering. (n.d.). *Saylor Academy*. Retrieved May 22, 2022, from https://learn.saylor.org/mod/page/view.php?id=32970

[22] (N.d.). *Futureuniversity.com*. Retrieved May 23, 2022, from https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf