Desarrollo de contenido

Unidad 1

Ingeniería de Software

Tecnología en Desarrollo de Software

IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

ORD No 74 de 2017 · VIGILADA MinEducación

www.iudigital.edu.co

Tabla de Contenido

Presentación general del curso

Al construir un software habitualmente se comenten errores, pero el objetivo final del ingeniero de software es minimizar al máximo dichos errores. Por tanto, en este curso aprenderás los conceptos generales y la importancia de los requerimientos; adicional, daremos un repaso por los diferentes modelos y métodos tradicionales para la construcción del software

Aprenderemos sobre las diferentes técnicas para las pruebas de software y así obtener un software de calidad y testeado, sin olvidar la documentación del diseño de aplicaciones importante en la creación de manuales y orientaciones posteriores, tanto para programadores como para el usuario final.

Al finalizar el curso contarás con la apropiación de los conocimientos necesarios para gestionar y liderar un proyecto de software, aplicando técnicas actuales con las cuales tú y el equipo de desarrollo podrán realizar y evaluar un software competitivo en el mercado.

Objetivo general:

Conocer los diferentes métodos, técnicas y procedimientos adecuados para el diseño de la arquitectura y ejecución de pruebas en un proyecto de software.

Objetivos específicos:

1. Reconocer las características de los sistemas para evaluar los contextos organizacionales, según el enfoque de la Teoría general de sistemas
2. Comprender el ciclo de vida de software y las diferencias entre las metodologías de desarrollo.
3. Realizar diseño de software orientado a objetos, aplicando diversas técnicas de análisis y con el suficiente detalle para ser implementado.
4. Utilizar las herramientas que permiten la documentación del software que se desarrolla.
5. indicar las técnicas adecuadas de pruebas de software empleando métodos y modelos adecuados.

Pregunta orientadora

¿Te has preguntado por qué tantas empresas de software no perduran en el tiempo?, ¿sabes qué papel cumple la ingeniería del software dentro de estas empresas? o, ¿por qué los proyectos informáticos son tan costosos cuando se pide una revisión general del software?

Los gastos operacionales crecen en todos los frentes, con la llegada de la tecnología muchos procesos operacionales y repetitivos se destinaron a sistemas de cómputo, minimizando la carga prestacional y operativa de las compañías.

Instrucciones de procedimientos y trabajo normalizados mal escritos, equipamientos indebidamente montados, planos y dibujos incorrectos, dispositivos de producción inadecuados y puestos de trabajo mal diseñados, son sólo ejemplos de algunos errores humanos. No es extraño encontrar componentes de programas de software fabricados con instrucciones antiguas o no estandarizadas, debido a que la información se ha retrasado en llegar al trabajador, y este en su afán de avanzar establece su propio ritmo desconociendo la cadena de producción. Un mal diseño y un fallo en el levantamiento de requisitos puede

retrasar todo el proceso de construcción del software, lo que lleva directamente a reelaboraciones, retrasos, variaciones, y ayudando al aumento del plazo en la ejecución del proyecto y del costo, tanto para el cliente como para la empresa.

Mapa del curso

Glosario
Unidad 1

A

- **Acceso directo**
  Es un icono que permite abrir más fácilmente un determinado programa o archivo.

- **Adaptabilidad**
  Un sistema adaptativo es aquel que puede cambiar a sí mismo o su entorno si su efectividad es insuficiente para alcanzar sus metas u objetivos actuales o futuros. Atributos del *software* que aprovechan la oportunidad para su adaptación a diferentes entornos especificados, sin aplicar otras acciones o medios que los previstos para este propósito del software considerado. La capacidad de un sistema de aclimatarse física y funcionalmente a un nuevo entorno operativo, con un grado mínimo de degradación del rendimiento de la capacidad. La capacidad del sistema para ser adecuado para un nuevo uso o propósito y para un nuevo rango o fuentes de variación. Según Jackson (2016), la adaptabilidad es un atributo de un sistema resistente.

- **ADSL**
  Asymmetric Digital SubscriberLine. Tecnología para transmitir información digital a elevados anchos de banda. A diferencia del servicio dial up, ADSL provee una conexión permanente y de gran velocidad. Esta tecnología utiliza la mayor parte del canal para enviar información al usuario, y sólo una pequeña parte para recibir información del usuario.

- **Ancho de banda (*bandwidth*)**
  Expresa la cantidad de datos que pueden ser transmitidos en determinado lapso. En las redes se expresa en bps.

- **AOL**
  América Online: proveedor de servicios de Internet de los Estados Unidos.

- **Apache**
  Servidor web de distribución libre. Fue desarrollado en 1995 y ha llegado a ser el más usado de Internet.

- **Árbol (*tree*)**
  Estructura de datos en la cual los registros son almacenados de manera jerárquica.

- **Archivo adjunto**

Archivo que acompaña un mensaje de e-mail. Es apropiado para el envío de imágenes, sonidos, programas y otros archivos grandes.

- Arquitectura lógica
  La arquitectura lógica de un sistema se compone de un conjunto de conceptos y principios técnicos relacionados que respaldan el funcionamiento lógico del sistema. Incluye una arquitectura funcional, una arquitectura de comportamiento y una arquitectura temporal.

B

- Backup
  Copia de seguridad. Se hace para prevenir una posible pérdida de información.
- Banner
  Gráfico generalmente rectangular, que se inserta en una página web. Puede tener carácter publicitario.
- Base de datos
  Conjunto de datos organizados de modo tal que resulte fácil acceder a ellos, gestionarlos y actualizarlos.
- Bug
  Bicho, insecto. Error de programación que genera problemas en las operaciones de una computadora.
- Byte
  Unidad de información utilizada por las computadoras. Cada byte está compuesto por ocho bits.

C

- Cable coaxial
  Es el tipo de cable usado por las compañías de televisión por cable para establecer la conexión entre la central emisora y el usuario. La compañía telefónica AT&T usó el cable coaxial para la primera conexión transcontinental en 1941. También se lo utiliza mucho en las conexiones de redes de área local (lan). Según el tipo de tecnología que se use, se lo puede reemplazar por fibra óptica.
- Cable-módem
  Módem que conecta una computadora con Internet a alta velocidad, por medio de una línea de TV por cable.
- Caché
  En un navegador, el caché guarda copias de documentos de acceso frecuente, para que en el futuro aparezcan más rápidamente.
- Cliente/servidor
  Este término define la relación entre dos programas de computación en el cual uno, el cliente, solicita un

servicio al otro, el servidor, que satisface el pedido.

- Cliente
  la organización o persona que recibe un producto o servicio.

- Cluster
  Grupo, racimo; agrupamiento. En la tecnología de las computadoras un cluster es la unidad de almacenamiento en el disco rígido. Un archivo está compuesto por varios clusters, que pueden estar almacenados en diversos lugares del disco.

- Comando (*command*)
  Instrucción que un usuario da al sistema operativo de la computadora para realizar determinada tarea.

- Comprimir
  Reducir el tamaño de un archivo para ahorrar espacio o para transmitirlo a mayor velocidad. Uno de los programas de compresión más populares de Windows es WinZip.

- Controlador
  Programa que comanda los periféricos conectados a la computadora.

- Cookie
  Pequeño archivo de texto que un sitio web coloca en el disco rígido de una computadora que lo visita. Al mismo tiempo, recoge información sobre el usuario. Agiliza la navegación en el sitio. Su uso es controvertido, porque pone en riesgo la privacidad de los usuarios.

- Costo del ciclo de vida (lCC)
  El costo total de implementación y propiedad de un sistema durante su vida útil. Incluye el costo de desarrollo, adquisición, operación, mantenimiento, soporte y, en su caso, eliminación.

- CPU: *Central Processing Unit*.
  Unidad central de procesamiento. Es el procesador que con- tiene los circuitos lógicos que realizan las instrucciones de la computadora.

- Criterio de evaluación
  Una característica utilizada para evaluar o comparar elementos del sistema, interfaces físicas, arquitecturas físicas, arquitecturas lógicas o cualquier elemento de ingeniería.

- Criterios de aceptación
  La especificación de adquisición en el contexto del acuerdo general, debe establecer claramente los criterios por los cuales el adquiriente aceptará la entrega del proveedor. Se puede utilizar una matriz de verificación para aclarar estos criterios.

D

- Disponibilidad
  La disponibilidad es la probabilidad de que un sistema reparable o elemento del sistema esté operativo en un momento dado, bajo un conjunto dado de condiciones ambientales.

- DNS: *Domain Name System*.
  Sistema de Nombres de Dominio. Método de identificación de una dirección de Internet. Según este método, cada computadora de la red se identifica con una dirección unívoca, la URL (Uniform Resource Locator), compuesta de grupos de letras separados por puntos. Esa dirección se obtiene subdividiendo todas las computadoras en grupos grandísimos llamados TLD (*Top Level Domain*) que son afines entre sí por alguna razón.

- Domain
  Dominio.

- Dominio
  Conjunto de caracteres que identifica la dirección de un sitio web.

- Download
  Descargar, bajar. Transferencia de información desde Internet a una computadora.

- Dpi: *dots per inch*
  Puntos por pulgada. En las impresoras, la calidad de la imagen sobre el papel se expresa en dpi.

- Driver
  Controlador

E

- Extranet
  Parte de una intranet de acceso disponible a clientes y otros usuarios ajenos a la compañía.

F

- FAQ: *Frequently-asked questions.*
  Las preguntas más frecuentes (y sus respuestas) sobre el tema principal de un sitio web.

- Fibra óptica
  Tecnología para transmitir información como pulsos luminosos a través de un conducto de fibra de vidrio. La fibra óptica transporta mucha más información que el cable de cobre convencional. La mayoría de las líneas de lar- ga distancia de las compañías telefónicas utilizan la fibra óptica.

- Fuente
  Variedad completa de caracteres de imprenta de un determinado estilo.

G

- Gateway
  Puerta, acceso, pasarela. Punto de enlace entre dos sistemas de re- des.

- GIF animado
  Variante del formato GIF. Se usa en la *WorldWideWeb* para dar movimiento a íconos y banners.

- Giga
  Prefijo que indica un múltiplo de 1.000 millones, o sea 109. Cuando se emplea el sistema binario, como ocurre en informática, significa un múltiplo de 230, o sea 1.073.741.824.

- Gigabit:
  Aproximadamente 1.000 millones de bits (exactamente 1.073.741.824 bits).

- Gigabyte (GB)
  Unidad de medida de una memoria. 1 gigabyte = 1024 mega- bytes = 1.073.741.824 bytes.

- Gigaflop
  Medida de velocidad de una computadora equivalente a 1.000 millones de operaciones de coma flotante por segundo.

H

- Herramienta de modelado
  Una herramienta como una aplicación de software que se utiliza para desarrollar modelos.

I

- Implementación
  El proceso que realmente produce los elementos del sistema de nivel más bajo en la jerarquía del sistema (estructura de desglose del sistema).

- Infraestructura
  Los activos, sistemas y redes, ya sean físicos o virtuales, cuya incapacidad o destrucción tendría un efecto debilitante sobre la Seguridad, la seguridad económica nacional, la salud o seguridad pública, o cualquier combinación de los mismos.

- Ingeniería de sistemas basados en modelos (mbse)
  La aplicación formal de modelado para soportar los requisitos del sistema, el diseño, el análisis, la verificación y las actividades de validación que comienzan en la fase de diseño conceptual y continúan durante todo el desarrollo, y las fases posteriores del ciclo de vida.

- Interfaz
  Un límite compartido entre dos unidades funcionales, definido por varias características pertenecientes

a las funciones, intercambios de señales físicas y otras características. Un componente de hardware o software que conecta dos o más componentes con el fin de pasar información de uno a otro. Para conectar dos o más componentes con el fin de pasar información de uno a otro.

L

- Linux
  Sistema operativo gratuito para computadoras personales derivado de Unix.

M

- Mac OS
  Sistema operativo de las computadoras personales y las workstations de Macintosh.
- Modelo analítico
  Modelo matemático en el que se cargan los datos para su análisis. (Turbante, et al 2010).
- Modelo
  Una expresión de una regularidad observada. Una representación de similitudes en un conjunto o clase de problemas, soluciones o sistemas. Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar esta solución un millón de veces, sin tener que hacerlo de la misma manera dos veces.
- Módem
  Modulador-demodulador. Dispositivo periférico que conecta la computadora a la línea telefónica.

N

- Navegador
  Programa para recorrer la *World Wide Web*. Algunos de los más conocidos son *Netscape Navigator*, *Microsoft Explorer, Opera y Neoplanet*.
- Net
  WorldWideWeb.
- Netiquette
  Conjunto de reglas de etiqueta tácitas dentro de Internet.

P

- Paquete (*packet*)
  La parte de un mensaje que se transmite por una red. Antes de ser enviada a través de Internet, la información se divide en paquetes.
- Proyecto

Un esfuerzo temporal realizado para crear un producto, servicio o resultado único. Un esfuerzo de desarrollo que consiste en actividades técnicas y de gestión con el propósito de diseñar un sistema. Esforzarse con criterios definidos de inicio y finalización para crear un producto o servicio de acuerdo con los recursos y requisitos especificados.

- Puerto serial
  Conexión por medio de la cual se envían datos a través de un so- lo conducto. Por ejemplo, el mouse se conecta a un puerto serial. Las computa- doras tienen dos puertos seriales: COM1 y COM2.

- Puerto
  En una computadora, es el lugar específico de conexión con otro dispositivo, generalmente mediante un enchufe. Puede tratarse de un puerto serial o de un puerto paralelo.

Q

- Query
  Consulta. Búsqueda en una base de datos.

R

- Realidad virtual
  Simulación de un medio ambiente real o imaginario que se puede experimentar visualmente en tres dimensiones. La realidad virtual puede además proporcionar una experiencia interactiva de percepción táctil, sonora y de movimiento.

- Red
  En tecnología de la información, una red es un conjunto de dos o más computadoras interconectadas.

S

- Sistema cerrado
  Un sistema que no tiene interacciones con su entorno.

- Sistema de ingeniería
  Un sistema abierto y concreto de elementos técnicos o sociotécnicos que es el foco de un ciclo de vida de SE. Sus características incluyen ser creado por y para las personas, tener un propósito y satisfacer las propuestas de valor de las partes interesadas clave cuando se consideran como parte de un contexto de sistema más amplio. Un sistema diseñado es un sistema diseñado o adaptado para interactuar con un entorno operativo anticipado para lograr uno o más propósitos previstos mientras se cumplen las restricciones aplicables.

- Socket

Un socket es el punto final de una conexión. Método de comunicación entre un programa cliente y un programa servidor en una red.

- Usuario

  Individuo o grupo que interactúa con un sistema o se beneficia de un sistema durante su utilización.

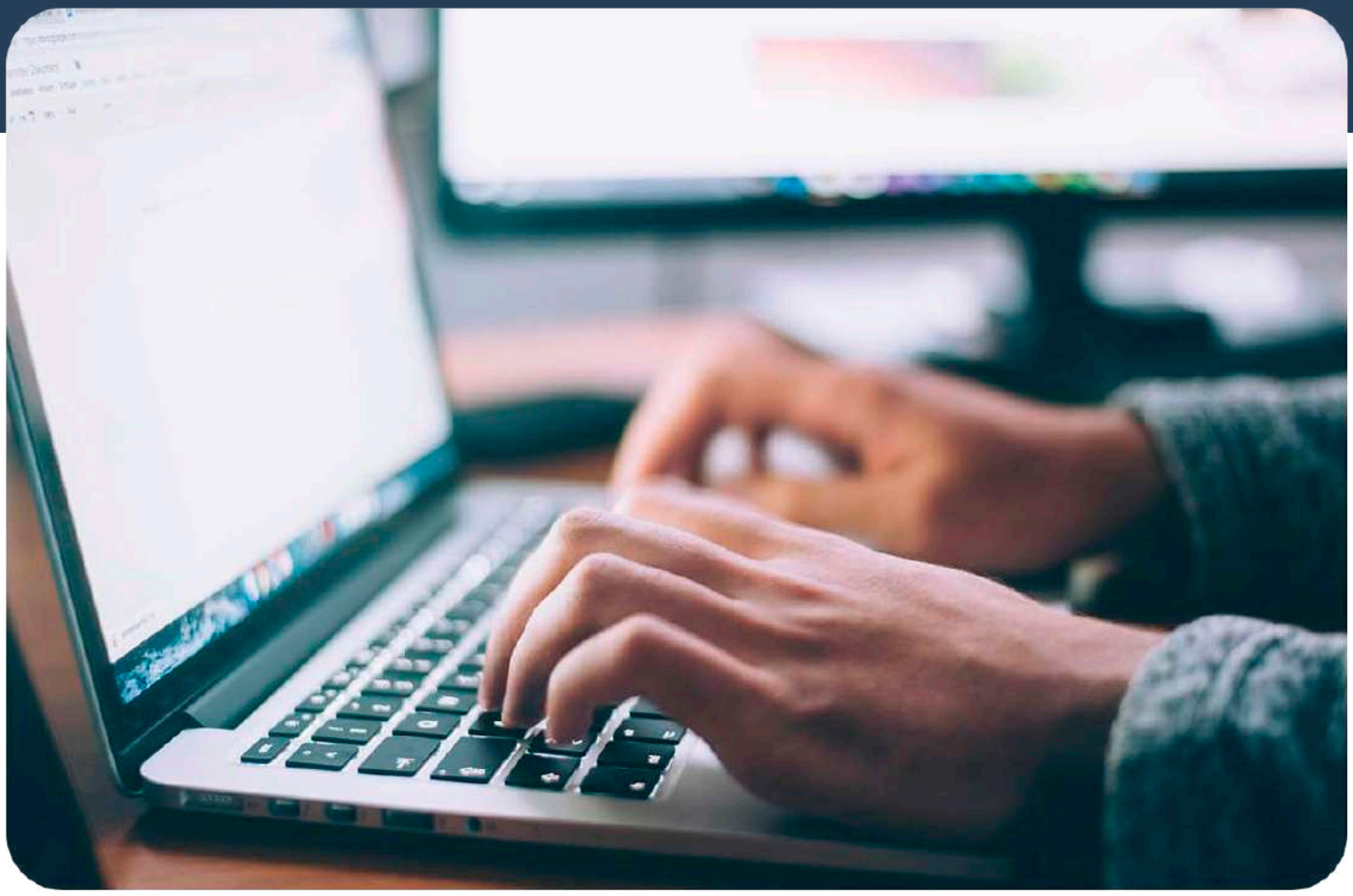
V

- Vida útil

  El período de tiempo durante el cual se espera que un activo o propiedad sea utilizable para el propósito que fue adquirido. Puede o no corresponder con la vida física real o la vida económica del artículo.

Unidad 1.

Introducción a la Ingeniería del Software



Introducción unidad 1

La ingeniería del software está relacionada con el diseño, instalación y mejoramiento de los programas de cómputo integrados principalmente en máquinas. ya sea de cómputo o de ayuda en la fabricación de productos. La ingeniería del software requiere de conocimientos en metodologías y métodos propios de diseño e implementación del software, proveniente de estándares ya probados y regulados por organizaciones y grandes industrias. La ingeniería del software se vale de los aportes de las ciencias exactas para poder diseñar, innovar, administrar y mejorar tales metodologías, las cuales se integran a cada proyecto con la finalidad de brindar

una guía para su elaboración. La unidad inicia con la definición de conceptos donde veremos algunas dificultades por resolver en la construcción del software, continuando con los requerimientos para finalizar con los modelos y métodos tradicionales de ciclo de vida del este.

Todo esto se enmarca en la importancia de ajustarnos a estándares y métodos ya diseñados en la construcción de software y no iniciar el proyecto sin un rumbo fijo de trabajo para su realización.

Actividad 1
conocimientos previos

Nombre de la actividad:
Análisis de información.

Objetivo de la actividad:
Analizar información teniendo como referencia los conocimientos previos sobre el tema de documentación del sistema, y el reconocimiento de los principales conceptos que se abordarán en la unidad 1.

Descripción:
Esta actividad está centrada en la discusión por medio de un foro, en este espacio debes responder a los interrogantes que encuentras más abajo. Recuerda que no es necesaria una consulta previa, participa teniendo como referencia tus conocimientos, estos planteamientos se abordarán en el desarrollo de la unidad.

Pregunta 1:
¿Por qué son importantes las políticas de construcción del software?

Pregunta 2:
¿Qué impactos positivos o negativos puede tener una mala gestión al elaborar el código de software?

Importante

Es necesario recordar que todos aquellos textos que sean consultados en diversas fuentes bibliográficas requieren de una estricta citación, se solicita para ello el uso de normas APA en su última versión.

Desarrollo los temas de la actividad de aprendizaje 1

1. TEMA 1.
Definición, Conceptos

La ingeniería de software la podemos definir como un estudio detallado de ingeniería para el diseño, desarrollo y mantenimiento de software. La ingeniería de software se introdujo para abordar los problemas de los proyectos de software de baja calidad. Los problemas surgen cuando un software generalmente excede los plazos, los presupuestos y los niveles reducidos de calidad. Asegura que la aplicación se construya de manera consistente, correcta, a tiempo y dentro del presupuesto, así como dentro de los requisitos. La demanda de ingeniería de software también surgió para satisfacer la inmensa tasa de cambio en los requisitos del usuario y el entorno en el que se supone que la aplicación está funcionando.

La ingeniería de software es la aplicación de un enfoque sistemático disciplinado y estable para el desarrollo, operación y mantenimiento del mismo.

Los problemas que han llevado al desarrollo de la práctica actual de ingeniería de software se enumeran a continuación. Estos problemas siguen siendo marcados hoy en día, especialmente para el desarrollo de este progrma a gran escala.

- Dificultad para cumplir los tiempos de entrega en los proyectos de software.
- Los altos costos para su desarrollo.
- Errores en el software entregado después de extensas pruebas.

- El alto costo en tiempo y esfuerzo en su mantenimiento.
- Problemas para medir el progreso durante el desarrollo y el mantenimiento del software.

Los problemas anteriores son el resultado del hecho de que la construcción de software es diferente del hardware, o a cualquier otro sistema de ingeniería y por tanto es importante entender algunas de sus características:

- El software no se fabrica en el sentido tradicional, ya que es intangible.
- Los sistemas de hardware se desgastan. El software no lo hace.
- La industria del software no tiene los manuales de diseño bien desarrollados.

Con el fin de abordar los problemas de desarrollo de software mencionados anteriormente, y para reducir el caos potencial de desarrollo de un producto intangible, se necesita un marco llamado **proceso de software** para las tareas que se requieren para la construcción de software de alta calidad. El modelo de proceso utilizado en un proyecto determinado dependerá de la naturaleza de este. Sin embargo, las siguientes actividades básicas son comunes a todos los modelos de proceso:

- **Especificación del software:** son los requisitos funcionales obtenidos del usuario.
- **Diseño e implementación de software:** es la producción de este sistema como producto.
- **Validación de software:** es la que garantiza que se cumplan las especificaciones del cliente.
- **Evolución del software:** es la modificación del sistema para satisfacer las necesidades continuas del cliente

El *Software Engineering Institute* (SEI) ha desarrollado criterios para evaluar la madurez de la capacidad de un modelo de proceso determinado en una organización. Cada proceso se evalúa de acuerdo con niveles de capacidad. Las directrices de integración de modelos de madurez de capacidad (CMMI) son bastante complejas y no todas las organizaciones de software pueden querer adoptarlas formalmente. Sin embargo, el espíritu de CMMI siempre debe incorporarse a la cultura, ya que la idea de ser CMMI es que el desarrollo de software debe abordarse de una manera organizada y disciplinada.

Si deseas comprender el modelo CMMI te invito a visualizar el siguiente video y a realizar un resumen personal que te permita reforzar los aprendizajes obtenidos:

Video

- **Título:** Software Engineering Coach. (2019). S01 - CMMI es Ágil. Punto [Archivo de video].
- **URL:** https://www.youtube.com/watch?v=NHUjIPe3vUM

2. TEMA 2.
Ingeniería de Requerimientos

Introducción

La ingeniería de requisitos consiste en comunicarse con el cliente. El objetivo es llegar a un acuerdo escrito que describa la función del software a desarrollar. El producto final de esta actividad suele llamarse especificación y constituye la base de todas las actividades de desarrollo que siguen. Resulta que las actividades de esta etapa del ciclo de vida del desarrollo se encuentran entre las más difíciles, pero quizás las más importantes.

El problema de desarrollar una comprensión clara de lo que el cliente necesita se ha convertido en un desafío clásico. Los recién llegados al campo a veces se sorprenden al saber que a los propios representantes del cliente les resulta difícil comunicar sus necesidades con claridad. Dada esta situación, los proyectos con frecuencia se encuentran cambiando requisitos a lo largo del período de desarrollo. Cuanto más tarde el proceso de desarrollo o levantamiento de un requisito, más difícil y caro se vuelve. Por lo tanto, lo que puede parecer un tiempo excesivo invertido en esta primera fase es, de hecho, una excelente inversión en reducción de riesgos para todo el proyecto. Muchas herramientas se utilizan en esta etapa para aumentar la confianza en la comprensión de cómo debe ser el sistema, incluyendo prototipos rápidos, escenarios de usuario y funciones / listas de características. Incorporar herramientas de modelado y diseño a esta fase no es inusual. No es de extrañar que esta fase se caracterice por largas reuniones entre el desarrollador y el cliente.

Especificación, análisis y validación

El enfoque se centra en definir las características operativas del software, y tiene tres objetivos principales:

- Describir los requisitos del cliente con el énfasis y el lugar en el que correrá el software.
- Definir la base para el diseño del software.
- Definir el sistema operativo que se puede utilizar para la validación del sistema, una vez completado el desarrollo del software.

Las siguientes subactividades se identifican con el análisis y el modelado de requisitos:

Análisis

La reutilización es un objetivo importante en el desarrollo de software, ya que reduce los costos, aumenta la confiabilidad y reduce el tiempo de desarrollo. El análisis de dominio es el proceso de identificación de patrones que se pueden reutilizar. Estos patrones pueden ser funciones o características comunes que tienen el potencial de un uso amplio en un dominio de aplicación. Un dominio de aplicación es una clase de problema, como financiero o espacial; sin embargo, cuanto más amplio sea la reutilización, mejor.

Validación

La validación del desarrollo del software se realiza por modelados, los cuales proporcionan parámetros específicos para realizar la validación:

- Modelado de datos.
- Modelado basado en escenarios.
- Modelado orientado al flujo.
- Modelado dinámico.

Modelado de datos

El modelado de análisis a veces comienza con la identificación de todos los objetos de datos que se van a procesar en el sistema y las relaciones entre estos objetos. El modelado de datos se utiliza para aplicaciones de grandes bases de datos y sistemas de información.

Modelado basado en escenarios

La participación del usuario final en un proyecto de software es fundamental para su éxito. El modelado basado en escenarios proporciona mecanismos para capturar información sobre cómo los usuarios finales desean interactuar con el sistema. UML proporciona compatibilidad para el desarrollo de escenarios de interacción que comienzan con la creación de casos de uso que describen un uso del sistema por un usuario final específico. La dinámica de estos casos de uso se puede representar en diagramas de actividad UML similares a los diagramas de flujo. Se pueden capturar interacciones más complejas en diagramas de carril de natación UML que pueden modelar actividades simultáneas.

Modelado orientado al flujo

Aunque no forma parte de UML, los diagramas de flujo de datos de entrada-proceso-salida (DFD) siguen siendo una herramienta de modelado de análisis muy popular, y se pueden usar para aumentar los diagramas UML. El flujo de datos en el sistema se puede modelar de forma jerárquica con DFD y diagramas de contexto de nivel superior que se están perfeccionando con DFD más detallados en niveles inferiores.

Modelado dinámico

Una vez establecidas las relaciones de datos estáticos y atributos, es útil crear modelos orales de comportamiento para representar la respuesta de los sistemas a eventos externos. Los casos de uso se pueden usar para identificar eventos, y los diagramas de secuencia UML se pueden usar para modelar cómo los eventos desencadenan transiciones de un objeto a otro.

Análisis orientado a objetos

El enfoque orientado a objetos (OO) para el análisis representa el último "cambio de paradigma" en la metodología de análisis, y es personificado por el lenguaje Java en la etapa de implementación. Algunas de las afirmaciones que han hecho popular este enfoque son las siguientes:

- Los clientes pueden entender los modelos OO sin conocimientos de programación, facilitando así las fases tempranas de comunicación.

- Los lenguajes OO promueven la reutilización del código y la productividad del programador.
- Los métodos de diseño y análisis de OO son complacientes a la hora de realizar cambios.

  El enfoque de OO se basa en el modelado del dominio problemático mediante clases y objetos.

- **Clase:** define los datos y las abstracciones de procedimiento para la información contenida y el comportamiento de alguna entidad del sistema.
- **Método:** representación de uno de los comportamientos de una clase.
- **Objetos:** instancia de una clase específica. Los objetos pueden heredar los atributos y las operaciones definidas para una clase. Las clases a veces se ilustran como cookie cutters, y los objetos asociados como cookies.

El objetivo del análisis orientado a objetos es el diseño de todas las clases y métodos asociados que son adecuados para el sistema que se está desarrollando.

El lenguaje de modelado unificado (UML) se ha desarrollado para la asignación de objetos y el desarrollo de sistemas orientados a objetos (OO). UML se ha convertido en un estándar de la industria para el desarrollo de OO y se explicará a profundidad en la unidad dos del curso.

Ingeniería arquitectónica y diseño

La actividad de diseño es el puente entre los requisitos de software y los modelos de análisis, y la construcción de productos entregables. El diseño es el proceso de producir la depuración para la codificación y las pruebas. También es la actividad que establece la calidad del software. Los resultados de las actividades de diseño son representaciones que pueden evaluarse en función de la calidad. La lista de atributos de calidad de software citados con frecuencia a veces se denomina FRUPS, un acrónimo de la siguiente lista:

- Funcionalidad.
- Fiabilidad.
- Usabilidad.
- Rendimiento.
- Compatibilidad.

Hay una gran diferencia entre simplemente conseguir que el código funcione y la ingeniería de un sistema de alta calidad. Los siguientes conceptos de diseño han sido útiles para lograr la calidad del software.

Abstracción

Al desarrollar un diseño o arquitectura de un sistema complejo, se necesitan muchos niveles de abstracción para describir el sistema. Los niveles más altos contienen menos detalles y los niveles más bajos proporcionan cada vez más información del sistema. Las abstracciones de procedimiento contienen instrucciones, pero suprimen las colas. Las abstracciones de datos hacen referencia a los objetos de datos y sus propiedades.

Modularidad

Cuando se trata de la complejidad, los psicólogos cognitivos nos dicen que los seres humanos podemos tratar sólo de cinco a nueve trozos de información a la vez. Por lo tanto, la estrategia de diseñar un sistema como una colección de módulos integrados es necesaria para la comprensión del sistema.

Ocultación de información

Muchos diseños arquitectónicos son posibles para un proyecto determinado. ¿Cómo se evalúa la calidad de modularización de un sistema particular? Aplicando el principio de información oculta en el desarrollo de una modularización, lo que aumenta la calidad, según lo definido por los atributos. La idea es definir los límites del módulo para que la información local esté encapsulada y oculta de su mundo exterior. Las interfaces de módulo están diseñadas para comunicar solo la información que es esencial para invocar la funcionalidad del módulo. La aplicación cuidadosa de este principio genera grandes dividendos durante las fases de prueba y el mantenimiento de la vida útil del sistema.

Independencia funcional

Se sabe que los módulos que están diseñados para ser funcionalmente independientes y tienen interfaces simples con el sistema restante, son más fáciles de desarrollar, probar y mantener. Dos criterios para evaluar la independencia son la cohesión y el acoplamiento, las medidas de unicidad de la función y la conectividad entre módulos, respectivamente. La alta cohesión y el bajo acoplamiento contribuyen a una mayor calidad.

Refinamiento

El proceso de diseño a veces se denomina un refinamiento descendente de la abstracción en el sistema de nivel superior a los niveles inferiores sucesivos, mediante la aplicación de los principios de modularización anteriores. Los módulos se crean mediante la descomposición jerárquica.

Refactorización

Esta actividad suele ser específica de los métodos ágiles. Se refiere al rediseño o reestructuración interna de un componente o subsistema de manera que mejore su calidad y rendimiento.

Diseño y reutilización de patrones

Al igual que con otras disciplinas de ingeniería más maduras, uno siempre debe abordar las decisiones de diseño con la mentalidad de que los patrones de diseño utilizados en el pasado deben ser considerados primero, en lugar de proceder con un diseño derivado de la singularidad de los requisitos del proyecto en particular. Si los patrones del pasado no parecen ser adecuados, la creación de otros nuevos deben ser el siguiente nivel de consideración y contribuir a la biblioteca de patrones para el uso de proyectos futuros. Los patrones de diseño van desde el nivel arquitectónico hasta el diseño de detalle de componentes.

Diseño de nivel de componente

Este nivel de diseño describe los datos estructurales, interfaces y algoritmos. El diseño a nivel de componente se puede representar en un lenguaje de programación, pero también se describen a menudo en alguna otra representación intermedia, como un lenguaje de diseño de programa (PDL) para el diseño de módulos convencionales y el lenguaje de restricción de objetos (OCL) en el mundo del diseño orientado a objetos.

Diseño de la interfaz de usuario

Un error común de los proyectos de software es pasar muy poco tiempo comunicándose con el usuario. Es fácil para los expertos en software caer en el subconsciente de "saber lo que es bueno para el usuario". Lo que puede parecer "claramente bueno para el usuario" es con demasiada frecuencia diferente desde la

perspectiva de la propia persona. El uso de escenarios de usuario y diseños de pantalla de prototipos muy tempranos e iterativos puede ayudar para asegurar que el usuario está siendo entendido. Se ha dicho que se debe planear en la construcción, teniendo buenas pautas, como son:

- Poner al usuario en control.
- Reducir la carga de memoria del usuario.
- Hacer que la interfaz sea consistente.

Herramientas para testear el software

En cuanto a herramientas para la ingeniería de requerimientos se puede decir que no existen de manera general, sino que son el conjunto de herramientas para testear y en esta unidad se hará una pequeña introducción para comprender el orden lógico de construcción de software. En la segunda unidad se realizará un estudio más profundo y detallado de dicho proceso.

De esta manera, después de codificar el sistema de software en un producto entregable, se utilizan estrategias de prueba para validar los requisitos del sistema. Las estrategias de prueba están diseñadas para detectar errores en el sistema. La depuración es el proceso de encontrar el origen de los errores para la corrección. Las pruebas exhaustivas no son prácticas.
Por lo tanto, no importa cuántas pruebas se hace, nunca se sabe con certeza si se han detectado todos los errores. Puesto que las pruebas son un proceso de detección de la presencia de errores, el proceso de prueba no puede garantizar la ausencia de todos los errores. Un alto porcentaje de los recursos del proyecto se gastan en la fase de prueba.

Las pruebas suelen realizarse en dos fases, primero a nivel de componente, a veces llamadas pruebas unitarias. Las pruebas unitarias son seguidas por pruebas de integración en las que se prueban grupos cada vez más grandes de componentes que culminan en el sistema total. Las pruebas unitarias suelen ser realizadas por el desarrollador, y las pruebas de integración por un grupo de pruebas independiente. Las estrategias de prueba para el software de diseño convencional difieren un poco de las de los sistemas orientados a objetos.

En la siguiente página se encuentra más información sobre técnicas de testeo de software

Para aprender más

- **Título:** Novoseltseva, E. (2017). Técnicas de testeo de software y herramientas [Entrada de blog]. Apiumhub.
- **URL:** https://apiumhub.com/es/tech-blog-barcelona/tecnicas-de-testeo-de-software/

En esta página se enuncian cuatro pruebas de testeo:

1. Prueba unitaria.
2. Pruebas de integración.
3. Pruebas funcionales.
4. Pruebas de rendimiento.

Pruebas unitarias y de integración en el software convencional: las pruebas unitarias se centran en las rutas de ejecución a través de la lógica del programa de componentes con el objetivo de maximizar la detección de errores por cobertura de trayecto, mientras que las pruebas de integración implican valores de entrada y salida.

Pruebas unitarias y de integración en el software orientado a objetos: las pruebas unitarias se realizan con clases, cuya definición implica no sólo la lógica interna del programa, sino también atributos y operaciones, así como la comunicación y la colaboración. Las operaciones deben probarse en el contexto de una clase.

Se definen dos enfoques comunes para las pruebas de integración de sistemas orientados a objetos, los que son basadas en subprocesos y los basados en el uso. El enfoque basado en subprocesos prueba el conjunto de clases que responden a una entrada o evento del sistema determinado. Las pruebas de uso comienzan probando clases que son relativamente independientes de todas las demás, y continúan en etapas con la adición de una capa de clases dependientes, hasta que se abarque todo el sistema.

Después de las pruebas unitarias y de integración, todo el sistema se prueba de acuerdo con los requisitos del cliente. Esta fase de prueba final suele llamarse prueba de validación, e incluye pruebas alfa y beta. Las pruebas alfa se realizan en el sitio del desarrollador y las pruebas beta se producen más tarde en los sitios de usuario. La versión final del software se programa una vez se han completado las pruebas beta.

Métricas de productos para software

El uso de medidas objetivas de productos de desarrollo de software como medida empírica de calidad es algo controvertido en la comunidad de ingeniería de este programa. Algunos dicen que nuestra falta de comprensión básica del software justifica retrasar el desarrollo y el uso de tales métricas. Sin embargo, hay muchas métricas disponibles para ayudar a evaluar y guiar el análisis, el diseño, el desarrollo del código fuente y las pruebas. A continuación, algunos ejemplos.

- **Análisis:** métrica de tamaño general del sistema, definida como una función de la información en el modelo de análisis.
- **Diseño:** métricas a nivel de componente que miden la complejidad.
- **Código fuente:** métrica de longitud definida en términos de líneas de código (LOC).
- **Pruebas:** cobertura del programa como gráfico dirigido.

3. TEMA 3.
Modelos y Métodos

Se han desarrollado varios modelos de procesos, cada uno hace hincapié en diferentes aspectos del ciclo de vida del software, y cada uno será apropiado según las características del proyecto.

Modelos de proceso de software

Una de las partes complejas de construir un sistema de software es decidir qué construir. Ninguna otra parte de la labor conceptual es tan difícil como establecer o recoger los requisitos detallados del usuario y asignarlos a las funciones y características del sistema. Ninguna otra parte tiene un impacto tan importante en los resultados posteriores y ninguna otra parte es tan difícil de rectificar.

Este proceso de recopilación de requisitos y su conversión en un producto "construido" ha sido abordado por muchos profesionales y ha dado como resultado una amplia variedad de modelos para el desarrollo de software.

Entre los modelos de desarrollos de software más populares tenemos:

- Modelo de cascada.

- Modelo en V.
- En flor.
- Prototipos.
- Modelo de espiral.
- Modelo de procesos.
- Desarrollo incremental.

Debido a que los pasos del ciclo de vida se describen en términos muy generales, los modelos son adaptables y sus detalles de implementación variarán entre las diferentes organizaciones. El modelo en espiral es el más general. De hecho, la mayoría de los modelos de ciclo de vida pueden derivarse como instancias especiales del modelo en espiral. Una Unidad de TIC puede mezclar y combinar diferentes modelos de ciclo de vida para desarrollar un modelo más adaptado a sus productos y capacidades.

Hay otros modelos que se pueden consultar, como el modelo basado en componentes, el modelo de especificaciones formales, el modelo concurrente y otros modelos que son propietarios. Estos métodos no se discuten en esta unidad porque son demasiado especializados o son adecuados para su uso en proyectos grandes y complejos.

El modelo en espiral es el que se utiliza en el segmento de aplicaciones de software y se recomienda su uso con el lenguaje de modelado unificado.

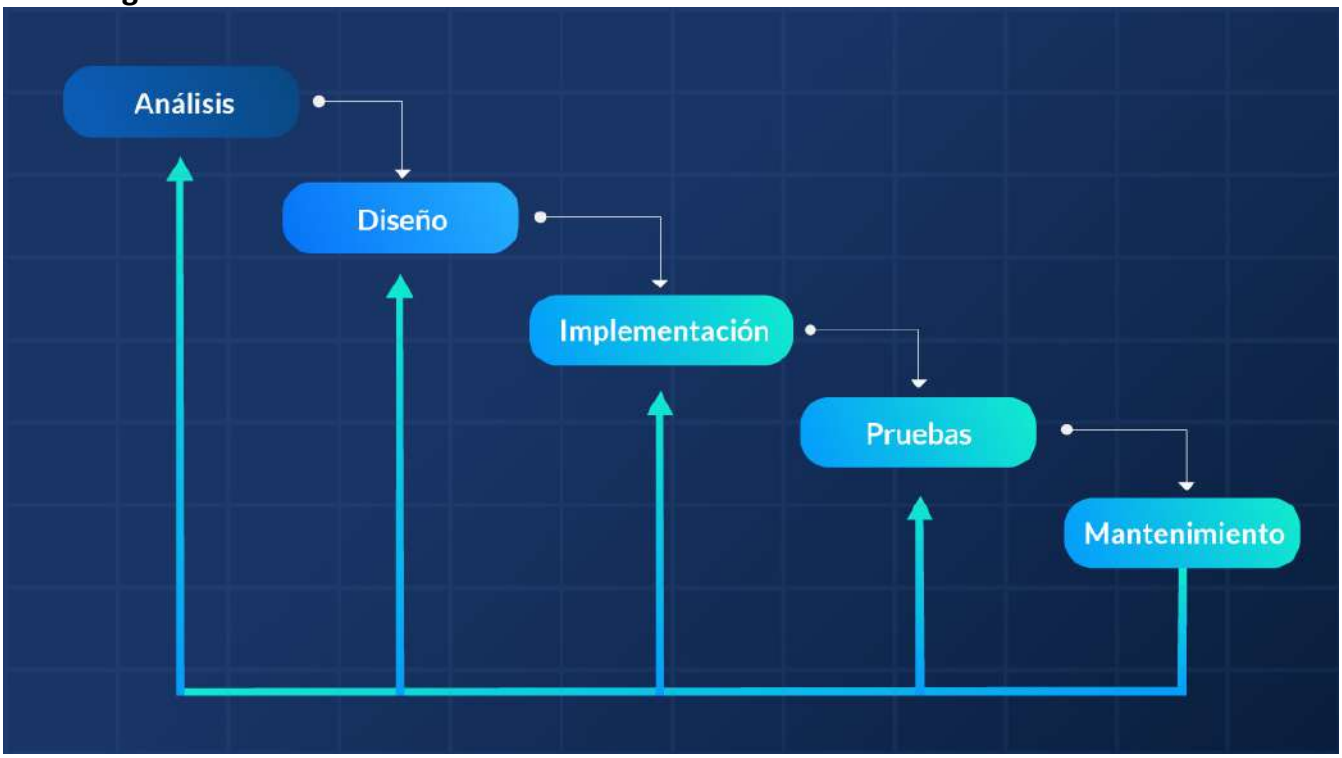
El UML se convierte en el vehículo de expresión de todas las actividades de desarrollo de software intelectual que llevan a los desarrolladores a la etapa de construir los productos utilizando plataformas de desarrollo interconectadas con la información del UML.

Modelo de Cascada

En 1970 Royce propuso como concepto inicial lo que actualmente se conoce como el modelo de cascada, un modelo que argumentó era defectuoso. Su artículo exploró cómo el modelo inicial podría desarrollarse en un modelo iterativo, con comentarios de cada fase que influyen en las fases posteriores. Es solo el modelo inicial el que recibió atención, por el contrario, su propia crítica de este modelo ha sido ampliamente ignorada. La frase "modelo de cascada" llegó rápidamente a referirse, no al diseño iterativo final de Royce, sino a su modelo puramente secuencial. Este apartado utiliza el significado popular de la frase "modelo de cascada". Para un modelo iterativo similar a la visión final de Royce, consulte sobre el modelo en espiral.

A pesar de las intenciones de Royce para que el modelo de cascada se modifique en un modelo iterativo, el uso del modelo de cascada como un proceso puramente secuencial sigue siendo popular y, para algunos, la frase "modelo de cascada" ha llegado a referirse a cualquier enfoque de software. Creación que se ve como inflexible y no iterativa. Aquellos que usan la frase "modelo de cascada" peyorativamente para los modelos no iterativos que no les gustan, generalmente ven el modelo de cascada en sí mismo como ingenuo e inadecuado para un proceso iterativo.

Figura 1. Modelo en cascada



Fuente: Soloriio, M. (2013). Metodología en cascada [Entrada de blog]. Recuperado de http://metodologiaencascada.blogspot.com/

En el "modelo de cascada" sin modificar, el progreso fluye de arriba a abajo, como una cascada. En el modelo de cascada original de Royce, las siguientes fases se siguen en orden:

- Especificación de requisitos.
- Diseño.
- Construcción (implementación AKA o codificación).
- Integración.
- Pruebas y depuración (validación AKA).
- Instalación.

- Mantenimiento.

Para seguir el modelo de cascada uno pasa de una fase a la siguiente de manera puramente secuencial. Por ejemplo, uno primero completa la especificación de requisitos, que se establecen en piedra. Cuando los requisitos están completamente completos, se procede al diseño. El software en cuestión está diseñado y se dibuja un anteproyecto para que lo implementen los programadores. Hacia las etapas posteriores de esta fase de implementación se integran componentes de software dispares producidos por diferentes equipos. Una vez completadas las fases de implementación e integración, el producto de software se prueba y depura; Las fallas introducidas en fases anteriores se eliminan aquí. Luego, el producto de software se instala y se mantiene para introducir una nueva funcionalidad y eliminar errores.

Por lo tanto, el modelo de cascada mantiene que uno debe pasar a una fase solo cuando su fase anterior se complete y se perfeccione. Las fases de desarrollo en el modelo de cascada son discretas y no hay saltos de un lado a otro ni se superponen entre sí.

Sin embargo, hay varios modelos de cascada modificados (incluido el modelo final de Royce) que pueden incluir variaciones leves o importantes en este proceso.

El Modelo en V

El modelo V es un modelo SDLC (Systems Development Life Cycle o Ciclo de vida de desarrollo de un sistema) donde la ejecución de procesos se realiza de forma secuencial en forma de V. También es conocido como modelo de verificación y validación.

El modelo V es una extensión del modelo de cascada y se basa en la asociación de una fase de prueba para cada etapa de desarrollo correspondiente. Esto significa que, para cada fase en el ciclo de desarrollo, hay una fase de prueba directamente asociada. Este es un modelo altamente disciplinado, y la siguiente fase comienza solo después de completar la fase anterior.

Bajo el Modelo V, la fase de prueba correspondiente de la fase de desarrollo se planea en paralelo. Por lo tanto, hay fases de verificación en un lado de la 'V' y fases de validación en el otro lado. La fase de codificación une los dos lados del V-Model.

La siguiente ilustración muestra las diferentes fases en un modelo V:

Figura 2. Modelo en V

Fuente: Weebly. (2010). Ciclo de vida en "V". Recuperado de: https://ingsoftware.weebly.com/ciclo-de-vida-en-v.html

Modelo V - Fases de verificación

Hay varias fases de verificación en el modelo V, cada una de ellas se explica en detalle a continuación.

Análisis de requerimientos de negocios

Esta es la primera fase del ciclo de desarrollo donde los requisitos del producto se entienden desde la perspectiva del cliente. Esta fase implica una comunicación detallada con el cliente para comprender sus expectativas y los requisitos exactos. Esta es una actividad muy importante y debe gestionarse bien, ya que la mayoría de los clientes no están seguros de qué necesitan exactamente. La planificación del diseño de la prueba de aceptación se realiza en esta etapa, ya que los requisitos comerciales se pueden utilizar como información para las pruebas de aceptación.

Diseño de sistemas

Una vez que tengas los requisitos de productos claros y detallados, es hora de diseñar el sistema completo. El diseño del sistema comprenderá y detallará la configuración completa de hardware y comunicación para el producto en desarrollo. El plan de prueba del sistema se desarrolla en base al diseño del sistema. Hacer esto en una etapa anterior deja más tiempo para la ejecución de la prueba real más adelante.

Diseño arquitectónico

Las especificaciones arquitectónicas se entienden y diseñan en esta fase. Generalmente se propone más de un enfoque técnico y, en función de la viabilidad técnica y financiera, se toma la decisión final. El diseño del sistema se divide en módulos que toman funcionalidades diferentes. Esto también se conoce como diseño de alto nivel (HLD).

La transferencia de datos y la comunicación entre los módulos internos y con el mundo exterior (otros sistemas), se entiende y define claramente en esta etapa. Con esta información, las pruebas de integración se pueden diseñar y documentar durante esta etapa.

Módulo de diseño

En esta fase se especifica el diseño interno detallado de todos los módulos del sistema, denominado Diseño de bajo nivel (LLD). Es importante que el diseño sea compatible con los otros módulos en la arquitectura del sistema y los otros sistemas externos. Las pruebas unitarias son una parte esencial de cualquier proceso de desarrollo, y ayudan a eliminar las fallas y errores máximos en una etapa muy temprana. Estas pruebas unitarias pueden diseñarse en esta etapa basándose en los diseños de los módulos internos.

Fase de codificación

La codificación real de los módulos del sistema, definidos en la fase de diseño se toma en la fase de codificación. El mejor lenguaje de programación adecuado se decide en función del sistema y los requisitos arquitectónicos.

La codificación se realiza en base a las pautas y estándares de codificación. El código pasa por numerosas revisiones de código y está optimizado para obtener el mejor rendimiento antes de que la compilación final se registre en el repositorio.

Fases de validación

Las diferentes fases de validación en un modelo V se explican en detalle, a continuación.

- Examen de la unidad

Las pruebas unitarias diseñadas en la fase de diseño del módulo se ejecutan en el código durante esta fase de validación. La prueba de unidad es la prueba a nivel de código y ayuda a eliminar errores en una etapa temprana, aunque no se pueden descubrir todos los defectos mediante la prueba de unidad.

- Pruebas de integración

Las pruebas de integración están asociadas a la fase de diseño arquitectónico. Las pruebas de integración se realizan para probar la coexistencia y comunicación de los módulos internos dentro del sistema.

- Pruebas del sistema

Las pruebas del sistema están directamente asociadas con la fase de diseño del sistema. Las pruebas del sistema verifican toda la funcionalidad del sistema y la comunicación del sistema en desarrollo con sistemas externos. La mayoría de los problemas de compatibilidad de software y hardware se pueden descubrir durante la ejecución de esta prueba del sistema.

- Test de aceptación

Las pruebas de aceptación están asociadas con la fase de análisis de requisitos de negocios, e involucran probar el producto en el entorno del usuario. Las pruebas de aceptación descubren los problemas de compatibilidad con los otros sistemas disponibles en el entorno del usuario. También descubre problemas no funcionales, como defectos de carga y rendimiento en el entorno de usuario real.

- Aplicación Modelo V

La aplicación del modelo V es casi la misma que la del modelo en cascada, ya que ambos modelos son de tipo secuencial. Los requisitos deben ser muy claros antes de que comience el proyecto, ya que generalmente es costoso volver atrás y hacer cambios.

Este modelo se utiliza en el campo del desarrollo médico, ya que es estrictamente un dominio disciplinado.

Los siguientes son algunos de los escenarios más adecuados para usar la aplicación V-Model.

- Los requisitos están bien definidos, claramente documentados y fijados.
- La definición del producto es estable.
- La tecnología no es dinámica y es bien entendida por el equipo del proyecto.
- No hay requisitos ambiguos o indefinidos.
- El proyecto es corto.

Pros y Contras del Modelo V

La ventaja del método V-Model es que es muy fácil de entender y aplicar. La simplicidad de este modelo también hace que sea más fácil de manejar. La desventaja es que el modelo no es flexible a los cambios y, en caso de que haya un cambio en los requisitos, que es muy común en el mundo dinámico de hoy, resulta muy costoso realizar el cambio.

Las ventajas del método V-Model son las siguientes:

- Este es un modelo altamente disciplinado y las fases se completan una a la vez.
- Funciona bien para proyectos más pequeños donde los requisitos se entienden muy bien.
- Simple y fácil de entender y usar.
- Fácil de manejar debido a la rigidez del modelo.
- Cada fase tiene entregables específicos y un proceso de revisión.

Las desventajas del método del modelo V son las siguientes:

- Alto riesgo e incertidumbre.
- No es un buen modelo para proyectos complejos y orientados a objetos.
- Modelo pobre para proyectos largos y en curso.
- No es adecuado para los proyectos donde los requisitos tienen un riesgo de cambio de moderado a alto.
- Una vez que una aplicación está en la etapa de prueba, es difícil volver atrás y cambiar una funcionalidad.
- No se produce ningún software que funcione hasta el final durante el ciclo de vida.

Modelo en Flor

Para el progreso de cualquier producto de software se realiza una cadena de tareas entre la idea original y el producto final. Un modelo de desarrollo establece la disposición en la que se harán los procedimientos para la fabricación del producto y poseer a si una mejor probabilidad de lo que se realizará. Si hablamos específicamente del modelo en flor básicamente se fundamenta en la distribución de una flor, para el cual todos los pétalos u hojas que contenga dicha organización serán un ciclo por realizar.

Sin embargo, todas las etapas se deben de desplegar al mismo tiempo para así conseguir que el sistema llegue a conseguir un producto final. Estas son algunas de las etapas más relevantes que debe poseer este modelo:

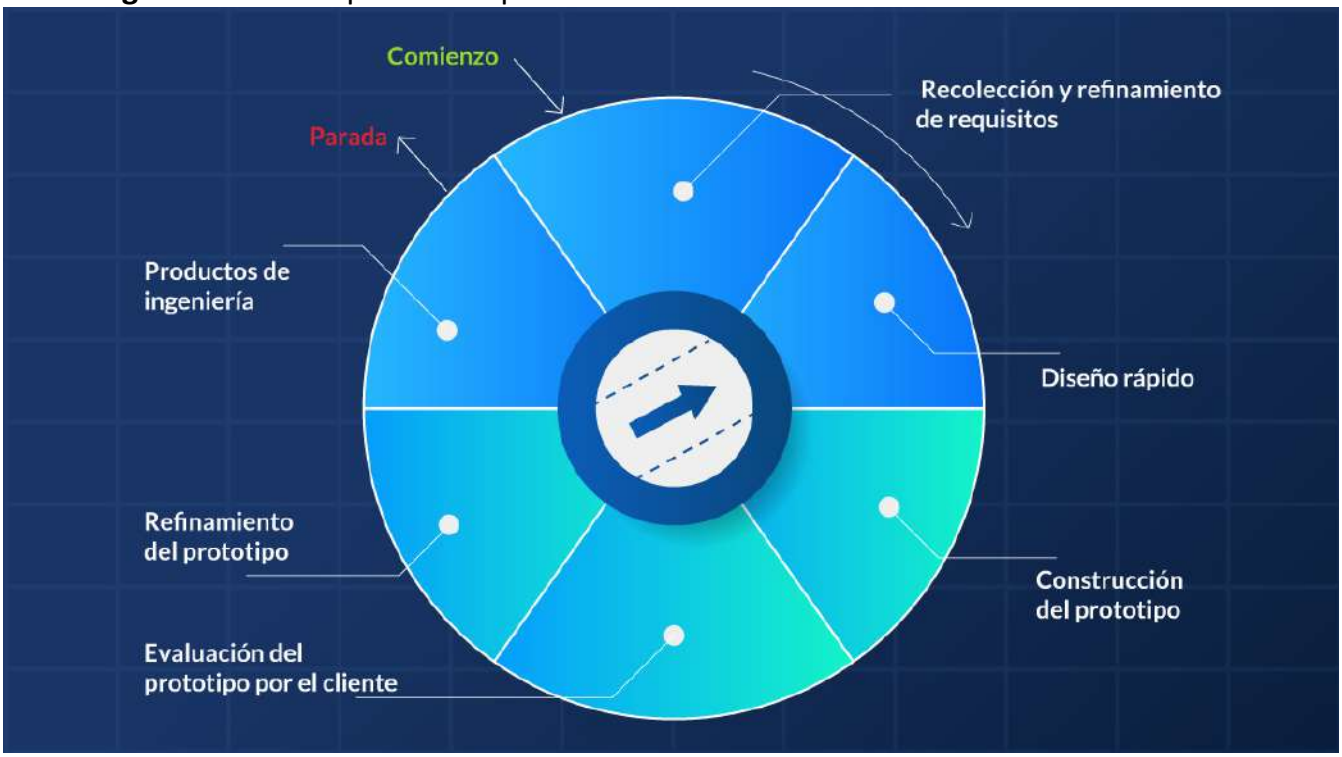
- Análisis.
- Diseño.
- Inicio.
- Código.
- Implementación.
- Pruebas.

Modelo por Prototipos

Los clientes a menudo definen un conjunto de objetivos generales para el software, pero no identifican los requisitos detallados de procesamiento o entrada.

El paradigma de prototipado ayuda al software y a que el cliente entienda mejor lo que se construirá cuando los requisitos sean confusos.

Figura 3. Modelo por Prototipos



Fuente: Ecured. (2011). Mprototipo. Recuperado de: https://www.ecured.cu/images/7/74/Mprototipo.png

El protocolo de creación de prototipos comienza con la comunicación donde se definen los requisitos y objetivos del software.

La iteración de prototipos se planifica rápidamente y se realiza el modelado en forma de diseño rápido.

- El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente "Interfaz humana".
- El diseño rápido conduce a la construcción del prototipo.
- El prototipo es desplegado y luego evaluado por el cliente.
- Los comentarios se utilizan para refinar los requisitos del software.

La iteración se produce cuando el prototipo se ajusta para satisfacer las necesidades del cliente, al tiempo que permite al desarrollador comprender mejor lo que se necesita hacer.

El prototipo puede servir como el "primer sistema". Tanto a los clientes como a los desarrolladores les gusta el paradigma de creación de prototipos, ya que los usuarios tienen una idea del sistema real y los desarrolladores crean software de forma inmediata. Sin embargo, la creación de prototipos puede ser problemática:

1. El cliente ve lo que parece ser una versión de trabajo del software, sin saber que el prototipo se mantiene unido "con goma de mascar". "Calidad, capacidad de mantenimiento a largo plazo". Cuando se informa de que el producto es un prototipo, el cliente se queja y exige que se apliquen algunas correcciones para convertirlo en un producto funcional. Muy a menudo, la gestión de desarrollo de software cede.

2. El desarrollador construye compromisos de ejecución a fin de obtener un prototipo de trabajo rápidamente. Un (Sistema Operativo) O / S inapropiado o lenguaje de programación utilizado simplemente (Costo Beneficio) b / c está disponible y es conocido. Después de un tiempo, el desarrollador puede sentirse cómodo con estas opciones y olvidarse de todas las razones por las que fueron inapropiadas.

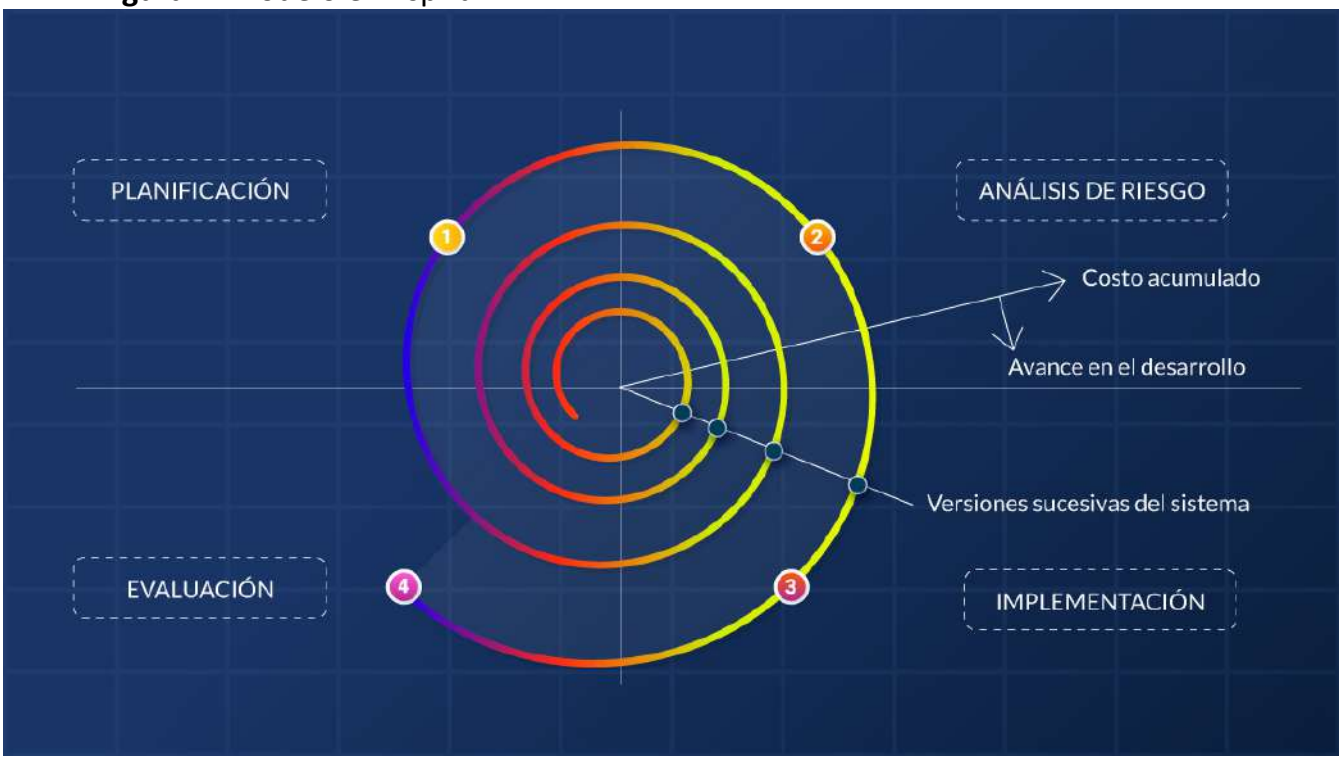
La clave es definir las reglas del juego al principio. El cliente y el desarrollador deben estar de acuerdo en que el prototipo está diseñado para servir como un mecanismo para definir los requisitos.

El Modelo de Espiral

El modelo espiral es uno de los modelos de ciclo de vida de desarrollo de software más importantes, que proporciona soporte para la gestión de riesgos. En su representación esquemática, parece una espiral con muchos bucles. Se desconoce el número exacto de bucles de la espiral y puede variar de un proyecto a otro. Cada ciclo de la espiral se llama Fase del proceso de desarrollo de software. El gerente del proyecto puede variar el número exacto de fases necesarias para desarrollar el producto, dependiendo de los riesgos del proyecto. Como el gerente determina dinámicamente el número de fases, este tiene un papel importante para desarrollar un producto usando un modelo en espiral.

El radio de la espiral en cualquier punto representa los gastos (costos) del proyecto hasta el momento, y la dimensión angular representa el progreso realizado hasta el momento en la fase actual. El siguiente diagrama muestra las diferentes fases del modelo espiral:

Figura 4. Modelo en Espiral



Fuente: José. (2019). Metodología de desarrollo de software (III) – Modelo en Espiral [Entrada de Blog]. Recuperado de https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/

Cada fase del modelo espiral se divide en cuatro cuadrantes, como se muestra en la figura anterior. Las funciones de estos cuatro cuadrantes se analizan a continuación:

Determinación de objetivos e identificación de soluciones alternativas: los clientes recopilan los requisitos y los objetivos se identifican, elaboran y analizan al comienzo de cada fase. Luego se proponen soluciones alternativas posibles para la fase en este cuadrante.

Identificar y resolver riesgos: durante el segundo cuadrante se evalúan todas las soluciones posibles para seleccionar la mejor solución posible. Luego se identifican los riesgos asociados con esa solución y los riesgos se resuelven utilizando la mejor estrategia posible. Al final de este cuadrante el prototipo está construido para la mejor solución posible.

Desarrolle la próxima versión del producto: durante el tercer cuadrante, las características identificadas se desarrollan y verifican mediante pruebas. Al final del tercer cuadrante, está disponible la próxima versión del software.

Revise y planifique la próxima fase: En el cuarto cuadrante, los clientes evalúan la versión desarrollada hasta ahora del software. Al final, se inicia la planificación para la siguiente fase.

Manejo de riesgos en modelo espiral

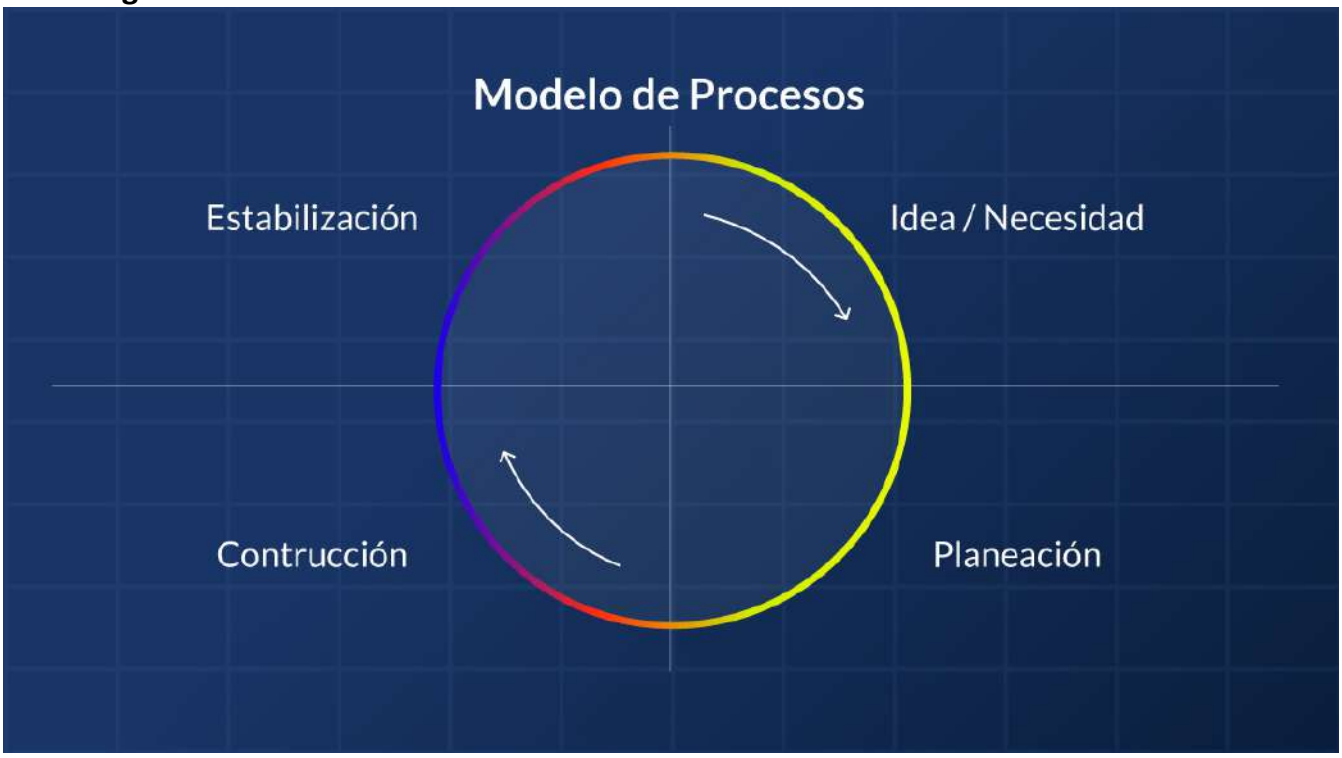
Un riesgo es cualquier situación adversa que pueda afectar la finalización exitosa de un proyecto de software. La característica más importante del modelo espiral es manejar estos riesgos desconocidos después de que el proyecto ha comenzado. Tales resoluciones de riesgos se hacen más fácilmente desarrollando un prototipo. El modelo en espiral permite hacer frente a los riesgos al proporcionar el alcance para construir un prototipo en cada fase del desarrollo del software.

El modelo de prototipos también admite la gestión de riesgos, pero los riesgos deben identificarse completamente antes del inicio del trabajo de desarrollo del proyecto. Pero en la vida real, el riesgo del proyecto puede ocurrir después de que comience el trabajo de desarrollo, en ese caso, no podemos usar el modelo de prototipos. En cada fase del Modelo Espiral, las características del producto fechado y analizado y los riesgos en ese momento se identifican y resuelven mediante la creación de prototipos. Por lo tanto, este modelo es mucho más flexible en comparación con otros modelos SDLC.

El Modelo de procesos

- Impulsa un proceso iterativo de desarrollo.
- Cada ciclo es una versión del producto.
- Utiliza metas definidas para marcar la transición entre las distintas etapas.
- Ofrece mayor poder de decisión a los usuarios.
- Busca mejorar la calidad y creatividad.

Figura 5. Modelo de Procesos



Fuente: elaboración propia

Las metas

Figura 6. Las Metas

Fuente: elaboración propia

Desarrollo Incremental

- Permite construir el proyecto en etapas incrementales en donde cada etapa agrega funcionalidad.
- Cada etapa consiste en requerimientos, diseño, codificación, pruebas, y entrega.
- Permite entregar al cliente un producto más rápido en comparación del modelo de cascada.
- Reduce los riesgos ya que:
  - Provee visibilidad sobre el progreso a través de sus nuevas versiones.
  - Provee retroalimentación a través de la funcionalidad mostrada.
  - Permite atacar los mayores riesgos desde el inicio.
  - Se pueden hacer implementaciones parciales si se cuenta con la suficiente funcionalidad.
  - Las pruebas y la integración son constantes.
  - El progreso se puede medir en periodos cortos de tiempo.
  - Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos.

Lecturas
y Material Complementario

Te invitamos a explorar el siguiente material para que amplíes los temas estudiados hasta este momento. Lee y analiza cada uno de ellos, pues contienen información que te será de gran ayuda en el desarrollo del curso.

- Novoseltseva, E. (2017). *Técnicas de testeo de software y herramientas* [Entrada de blog]. Apiumhub. Recuperado de https://apiumhub.com/es/tech-blog-barcelona/tecnicas-de-testeo-de-software/
- UCAM Universidad Católica de Murcia. (2015). *Ingeniería de Requisitos - Conceptos básicos de la Ingeniería de Requisitos - Raquel Martínez España* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=AbwwtjOfJIY
- Wilson Agudelo. (2016). *Modelos de desarrollo de software* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=MJzBZ92nZ7E

Actividad
de Aprendizaje 1. Importancia de la ingeniería de software

Objetivo de la actividad:
Comprender la importancia de la ingeniería del software como parte indispensable en la calidad del producto final, comprendiendo el ciclo de vida de software y las diferencias entre las metodologías de desarrollo.

Descripción de la actividad
Esta actividad es de carácter individual y tiene por finalidad que compares la posición de tres autores que hablen sobre ingeniería de software. Para lograr esto, realiza un rastreo de textos, artículos científicos o documentos académicos que traten sobre la importancia de la ingeniería del software para la construcción de productos, el ciclo de vida del software y las diferencias entre las metodologías de desarrollo. Recuerda que este rastreo o consulta la puedes realizar a través de la base de datos académica de la IU Digital (https://ebookcentral.proquest.com/auth/lib/iudasp/login.action) o bases de uso libre como Redalyc (https://www.redalyc.org/) o Dialnet (https://dialnet.unirioja.es/).

Una vez hayas seleccionado los tres textos o documentos, realiza un cuadro comparativo evidenciando los elementos comunes y diferentes entre los autores. Al final del cuadro brinda dos o más conclusiones sobre la importancia de la ingeniería de software.

Recuerda que un cuadro comparativo es una herramienta o técnica de estudio que te permite encontrar similitudes y diferencias entre dos o más términos, conceptos o autores. Te invito a revisar la siguiente página en la que se te muestra cómo se construye un cuadro comparativo:

Cuadro Comparativo. (2019). Cuadro Comparativo: Qué es, tipos, características, ejemplos. Recuperado de: https://cuadrocomparativo.org/cuadro-comparativo/

Evidencia:

Documento Word o Pdf, con letra Arial, tamaño 11 y espaciado sencillo. El documento debe contener:

- Hoja de presentación.

- Introducción.
- Cuadro comparativo de los elementos comunes y diferentes de mínimo tres autores.
- Conclusiones.
- Normas APA, última edición.

Actividad
de Aprendizaje 2. Modelo de ingeniería tradicional

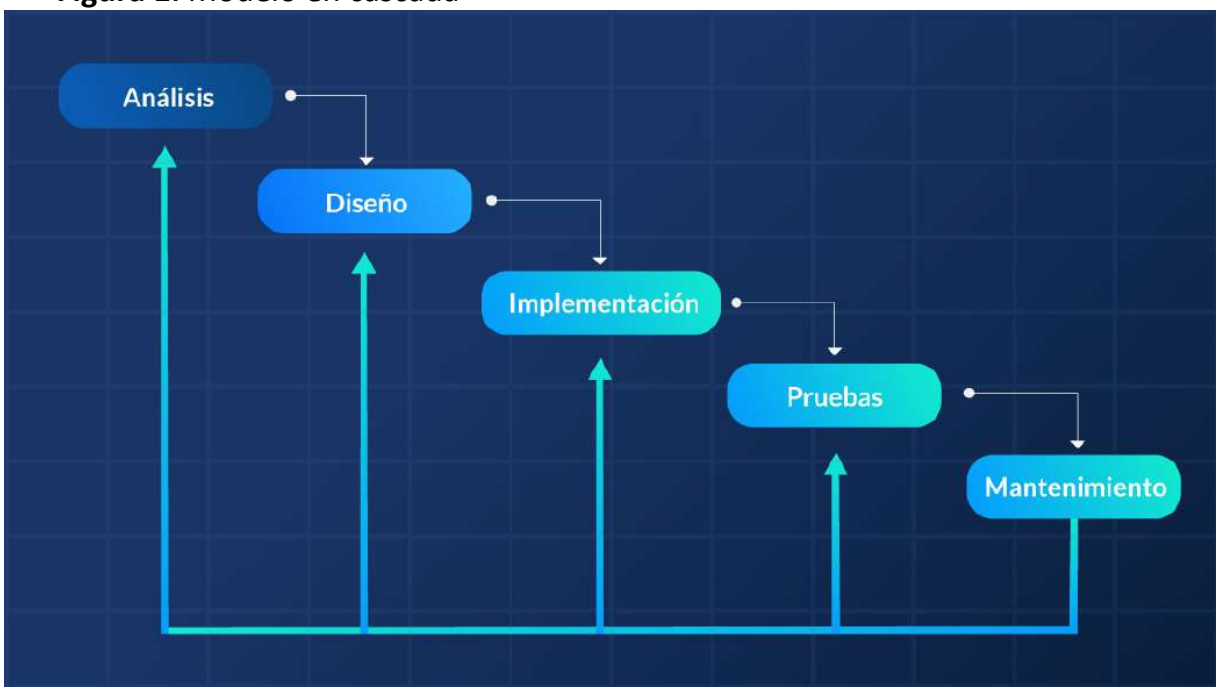
Objetivo de la actividad:
Comprender el ciclo de vida del software y las diferencias entre las metodologías de desarrollo a partir de la identificación de las etapas del modelo en cascada para la construcción de un proyecto de software.

Descripción de la actividad
Esta actividad es de carácter grupal y tiene como finalidad la identificación del modelo en cascada y sus diferentes etapas en el ciclo de vida tradicional para construcción de software. Para ello, selecciona un problema cotidiano que consideres que requiera el diseño de un software, por ejemplo, en una escuela quieren mejorar la comprensión lectora y el docente encargado quiere hacer un juego interactivo.

Una vez identificado el problema, aplica los pasos de la metodología del modelo en cascada teniendo en cuenta los siguientes pasos:

1. Recuerda y aplica los conocimientos adquiridos en otros cursos de la tecnología de ingeniería de software.
2. Realiza una nueva lectura de la unidad y el material complementario que se ha trabajado hasta el momento.
3. Revisa la rúbrica de evaluación para que tengas claro las categorías que se tendrán en cuenta para evaluar la actividad.
4. Crea y diligencia una tabla similar a la que encontrarás a continuación con el ejemplo, teniendo en cuenta cada una de las etapas del modelo en cascada que se referencian en la figura 1.

Figura 1. Modelo en cascada



Fuente: Soloriio, M. (2013). Metodología en cascada [Entrada de blog]. Recuperado de http://metodologiaencascada.blogspot.com/

Modelo en cascada			
Problema identificado		Una escuela quiere mejorar la comprensión lectora y el docente encargado quiere hacer un juego interactivo.	
Etapa		**Descripción de la etapa**	**Tareas o acciones según el problema identificado (mínimo 5 por etapa)**
1	**Análisis**	En la fase de análisis se evalúan los requerimientos y se realiza un estudio de viabilidad que …	1.Definición de la población a la cual va dirigida el juego interactivo. 2.Definición de tipo de interactividad que desea el cliente. 3. 4. 5.
2	**Diseño**		
3	**Implementación**		
4	**Pruebas**		
5	**Mantenimiento**		

Adicional te comparto el siguiente enlace en el cual puedes encontrar una explicación detallada del modelo en cascada:

Pedro Montecinos G. P.,(Abril 2018). Recuperado de: https://es.slideshare.net/pmontecinos/modelo-en-cascada-pemo

Evidencia:

Documento Word o Pdf, con letra Arial, tamaño 11 y espaciado sencillo. El documento debe contener:

- Hoja de presentación.
- Introducción.
- Definición del problema a resolver con la metodología en cascada.
- Descripción de la ejecución de cada un0 de los pasos del modelo en cascada
- Normas APA, última edición.

Es necesario recordar que todos aquellos textos que sean consultados en diversas fuentes bibliográficas requieren de una estricta citación, se solicita para ello el uso de normas APA en su última versión.

Proyecto Integrador:

A continuación, se describen los elementos a desarrollar en el planteamiento del proyecto:

Nombre: Ciclo de vida y documentación del software

Objetivo: identificar las etapas de un modelo de ciclo de vida clásico para la construcción de software, a partir de un desarrollo elaborado previamente y la documentación de este.

Descripción

Este proyecto se debe realizar de manera colaborativa y deben tener en cuenta los siguientes pasos:

1. Elijan un modelo de ciclo de vida clásico para la construcción del software (Cascada, Modelo en V, Prototipos o Espiral, entre otros).
2. Seleccionen uno de los ejercicios de programación realizados en cursos anteriores, procure que sea un ejercicio que recuerden cómo lo desarrollaron y qué pasos siguieron durante su diseño.
3. Una vez seleccionado el ejercicio de programación y el modelo de ciclo de vida, relaciónenlos identificando y describiendo cada una de las etapas del modelo. Para ello se pueden basar y enriquecer la matriz de la actividad 2 de la unidad 1.
4. Escriban las conclusiones del ejercicio señalando las ventajas y desventajas del modelo de vida clásica seleccionado.
5. Realicen la documentación del código del software o del manual de uso de usuario final.

Momentos o fases

Fase 1. Elección de modelo de ciclo de vida para construcción de software y ejercicio de programación.

Fase 2. Identificación de cada una de las etapas del modelo de ciclo de vida para la construcción de software en el ejercicio de programación desarrollado en un curso anterior, junto con su respectiva descripción.

Fase 3. Redacción de conclusiones señalando ventajas y desventajas del modelo de ciclo de vida para la construcción de software.

Fase 4. Elaboración de documentación del código del software o del manual de usuario final.

Evidencia

Documento Word o Pdf, con letra Arial, tamaño 11 y espaciado sencillo. El documento debe contener:

- Hoja de presentación.
- Introducción.
- Descripción de cada una de las etapas del modelo de ciclo de vida para la construcción de software a partir del ejercicio de programación desarrollado en un curso anterior.
- Documentación del código o del manual de usuario final.
- Normas APA, última edición.

Bibliografía

- Salvado J., (2019). 1. *En Desarrollo de Software Seguro*(66). https://www.lulu.com/shop/jose-salvador-gonzalez-rivera/auditor%C3%ADa-de-sistemas-de-gesti%C3%B3n/paperback/product-24215538

- Salvado J.. (2019). 2. *Auditoría de Sistemas de Gestión* (48). http://www.lulu.com/shop/jose-salvador-gonzalez-rivera/auditor%C3%ADa-de-sistemas-de-gesti%C3%B3n/paperback/product-24215538

- Pressman, Roger S, (2005), *Ingeniería de software: un enfoque profesional,* 6a Edición, McGraw-Hill

- INTECO (2009). *Introducción a la Ingeniería del Software*. Recuperado de https://openlibra.com/es/book/introduccion-a-la-ingenieria-del-software

- Sommerville, I. Addison-Wesley, (2004*) Ingeniería de software, 7a Edición*

- Thayer, R. y Dorfman, M. (Editores), (2002), *Ingeniería de software Volumen 1 - El proceso de desarrollo*, 2ª edición, IEEE Computer Society Press

- Thayer. R. y Christensen, M. (Editores), (2002*), Ingeniería de software Volumen 2 - El proceso de soporte, 2ª edición*, IEEE Computer Society Press

- Viega, John y McGraw, Gary, (2002) Creación de software seguro: cómo evitar problemas de seguridad de la manera correcta, Addison-Wesley

- Swiderski, Frank y Snyder, (2004) *Ventana, Modelado de Amenazas*, Microsoft Press

- Howard, Michael y LeBlanc, David, (2003) Escribiendo Código Seguro*, MicrosoftPress, Segunda edición*

- Schneier, Bruce, (2002), Secretos y Mentiras: Seguridad Digital en un Mundo en Red, JohnWiley and Sons, Inc.

- Kuperman, Benjamin A., Brodley, Carla E., Ozdoganoglu, Hilmi, Vijaykumar, TN y Jalote, Ankit,(2005), *Detección y prevención del desbordamiento de búfer de pila*, Comunicaciones de la ACM, 48, (noviembre) 51- 56

IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

ORD No 74 de 2017 · VIGILADA MinEducación

www.iudigital.edu.co