Instituto Superior Tecnológico Particular

"Bolívar Madero Vargas"

INGENIERÍA DE SOFTWARE I

PRINCIPIOS, CONCEPTOS, MÉTODOS Y HERRAMIENTAS

Guía de Estudio

Resumen del Libro de Roger Pressman

Docente: Ing. Andrea Cartuche Jara

Nivel: IV

Carrera: Tecnología en Desarrollo de Software

Código: GE- IS_4_DS1

Fecha de Elaboración: 10 de abril del 2023

Año lectivo

2023-2024

Contenido

GUIA DE ESTUDIO DE LA ASIGNATURA INENIERÍA DE SOFTWARE I

1. Introducción

Propósito de la guía

Esta guía de estudio ha sido elaborada con el objetivo de proporcionar un resumen estructurado del libro *Ingeniería de Software* de Roger Pressman. Se busca facilitar la comprensión de los conceptos fundamentales de la disciplina, abordando tanto los principios teóricos como las mejores prácticas aplicadas en el desarrollo de software.

2. Importancia de la asignatura en el plan de estudios

La Ingeniería de Software es una disciplina clave en el desarrollo de sistemas computacionales eficientes, escalables y de alta calidad. En el contexto del plan de estudios de Tecnología en Desarrollo de Software, esta asignatura proporciona las bases metodológicas y técnicas necesarias para diseñar, construir y mantener productos de software con estándares profesionales. Su importancia radica en:

- Mejorar la capacidad de análisis y diseño de sistemas.
- Garantizar la calidad del software mediante metodologías y procesos bien definidos.
- Facilitar el trabajo en equipo y la gestión de proyectos de software.
- Promover el uso de buenas prácticas en el ciclo de vida del desarrollo de software.

3. Objetivos generales y específicos

Objetivo general

Brindar a los estudiantes un marco de referencia sólido sobre los principios, modelos y metodologías de la Ingeniería de Software, basado en el libro de Roger Pressman, para que puedan aplicarlos en el desarrollo y mantenimiento de proyectos de software.

Objetivos específicos

1. Comprender los fundamentos de la Ingeniería de Software y su evolución a lo largo del tiempo.
2. Identificar y analizar las distintas metodologías y modelos de desarrollo de software.
3. Aplicar buenas prácticas en la gestión de proyectos de software, desde la planificación hasta la implementación.
4. Evaluar la calidad del software mediante métricas y procesos de aseguramiento.
5. Desarrollar una mentalidad crítica y analítica para la solución de problemas en el ámbito del software.

INGENIERÍA DE SOFTWARE I

- ¿Qué es? El software de computadora es el producto que construyen los programadores profesionales. La ingeniería de software está formada por:
  - Un **proceso**
  - Un conjunto de **métodos** (prácticas); y
  - Un arreglo de **herramientas** que permite a los profesionales elaborar software de cómputo de alta calidad.

IMPORTANCIA DE LA INGENIERÍA DE SOFTWARE

La ingeniería de software es importante, porque nos permite construir sistemas **COMPLEJOS** en un **TIEMPO** razonable y con **ALTA CALIDAD.**

- **SISTEMAS COMPLEJOS**
- **TIEMPO RAZONABLE**
- **ALTA CÁLIDAD**

ENFOQUE DE LA INGENIERÍA DE SOFTWARE

La aplicación de un proceso ágil y adaptable para obtener un resultado de mucha calidad, que satisfaga las necesidades de las personas que usarán el producto.

- ¿Cuál es el producto final? Desde el punto de vista del ingeniero es el conjunto de programas, contenido (datos) y otros productos terminados que constituyen el software de computadora. Desde la perspectiva del usuario, es la información resultante que hace mejor al mundo en el que vive.

Comentario: Ya sabes lo que se dice… puedes tenerlo rápido o bien hecho o barato. Escoge dos de estas características, Elijo RÁPIDO Y BIEN HECHO, ¡¡¡entonces necesitas Ingeniería de Software!!!

Conforme ha aumentado la importancia del Software, la comunidad de programadores ha tratado continuamente de desarrollar tecnologías que hagan más fácil, rápida y barata la elaboración de programas de cómputo de alta calidad.

LA ESTRUCTURA INCLUYE UN PROCESO, UN CONJUNTO DE MÉTODOS Y UNAS HERRAMIENTAS A LAS QUE LLAMAMOS INGENIERÍA DE SOFTWARE.

LA NATURALEZA DEL SOFTWARE

El Software es un **transformador de información**:

- Produce
- Administra
- Adquiere
- Modifica
- Despliega; o
- Transmite

información que puede ser tan simple como un solo bit o tan compleja como una presentación en multimedia.

El Software distribuye el **PRODUCTO** más importante de nuestro tiempo: **INFORMACIÓN**

El papel del software de cómputo ha sufrido un cambio significativo, las notables mejoras en el funcionamiento del hardware, los profundos cambios en las arquitecturas de computadora, el gran incremento en la memoria y capacidad de almacenamiento y una amplia variedad de opciones de entradas y salidas exóticas han propiciado la existencia de sistemas basados en computadora más sofisticados y complejos.

PREGUNTAS QUE CUANDO SE CONSTRUYEN SISTEMAS MODERNOS BASADOS EN CMPUTADORAS.

- ¿Por qué se requiere tanto **TIEMPO** para terminar el software?
- ¿Por qué son tan altos los **COSTOS** de desarrollo?
- ¿Por qué no podemos detectar todos los **ERRORES** antes de entregar el software a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a **MANTENER** los programas existentes?
- ¿Por qué seguimos con dificultades para **MEDIR EL AVANCE** mientras se desarrolla y mantiene el software?

CARACTERÍSTICAS DEL SOFTWARE

Para asimilar lo anterior, es importante examinar las características del software que lo hacen diferente de otros objetos que construyen los seres humanos.

EL SOFTWARE ES ELEMENTO DE UN SISTEMA LÓGICO Y NO DE UNO FÍSICO.

1. **El software se desarrolla o modifica con intelecto no se manufactura en el sentido clásico. -** La alta calidad se logra a través de un buen diseño, los costos del software se concentran en la ingeniería. Esto significa que los proyectos de software no pueden administrarse como si fueran proyectos de manufactura, es la construcción de un producto lógico.

2. **El Software no se desgasta. -** El hardware suele presentar defectos de diseño o manufactura como suciedad, vibración, abuso, temperaturas extremas, etc. El software no se desgasta, pero si se DETERIORA, se deteriora como consecuencia del CAMBIO, el hardware se desgasta y es sustituido por una refacción, en cambio no hay refacciones para el software.

3. **Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del Software se construye para un uso individualizado. -** Los componentes estandarizados para el diseño se utilizan mucho en cuanto a hardware, un componente de software debe diseñarse e implementarse de modo que pueda volverse a usar en muchos programas diferentes.

DOMINIOS DE APLICACIÓN DEL SOFTWARE

Hay siete grandes características:

1. Software de Sistemas: Conjunto de programas escritos para dar servicio a otros programas ejm. Compiladores, editores y herramientas para administrar archivos, software de redes.
2. Software de Aplicación (a medida): Programas aislados que resuelven una necesidad especifica de negocios, facilita las operaciones de negocio, se usa para controlar funciones de negocio en tiempo real.
3. Software de Ingeniería y Ciencias: Van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital, el diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a hacerse en tiempo real e incluso han tomado características del software de sistemas.
4. Software Incrustado: Reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. Por ejemplo, control de tablero de un horno microondas, control de funcionamiento digital de un automóvil, etc.
5. Software de Línea de Productos (paquetes aplicativos): Es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. Ejm. Procesamiento de textos, hoja de cálculo, multimedios, entretenimiento, administración de bases de datos, (office, Photoshop, etc.)
6. Aplicaciones Web: Llamadas WebApps, esta categoría de Software centrado en redes agrupa una amplia gama de aplicaciones, proveen características aisladas, funciones de cómputo y contenido para el uso final, también están integradas con bases de datos corporativas y aplicaciones de negocios.
7. Software de Inteligencia Artificial: Hace usos de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo, incluyen robótica, sistemas expertos, redes neuronales artificiales.

¿QUE ES LA CONVERSIÓN DE INFORMACIÓN?

Transformación de un esquema de representación de los elementos de información a otro: incluye estos elementos.

- Datos ingresados
- Reportes generados
- Parámetros o configuración de sistemas
- Formularios

Fusiones. - Poseen sistemas o estructuras de información distintas

Migración de Sistemas. - Sea por actualización de versiones o por migración a un nuevo sistema de información.

Adaptación
- Nuevos Procesos
- Nuevas Tecnologías
- Normas, leyes u otras exigencias

MIGRACIÓN DE SISTEMAS COMPUTACIONALES

Durante el ciclo de un software, puede surgir la necesidad de modificarlo para ejecutarse en un entorno diferente.

Se puede definir a la MIGRACIÓN DE SISTEMAS como: el conjunto de actividades para ACTUALIZAR, MODIFICAR O ELIMINAR EQUIPOS INFORMÁTICOS Y RECURSOS RELACIONADOS a la tecnología y sistemas de información incluyendo el área de telecomunicaciones.

Tiene por finalidad el traslado del sistema a un nuevo ambiente operativo, **CONSERVANDO SU FUNCIONALIDAD Y DATOS ORIGINALES.**

CAUSAS PARA REALIZAR MIGRACIÓN

- No existe soporte para el sistema actual
- No son suficientes las capacidades del hardware actual
- El mantenimiento del sistema actual resulta ser muy costoso
- El sistema actual puede no ser compatible con nuevas tecnologías que se desee incluir.

SOFTWARE HEREDADO

Estos programas antiguos, que es frecuente denominar software heredado, estos fueron desarrollados hace varias décadas y han sido modificados de manera continua para que satisfagan los cambios en los requerimientos de los negocios y plataformas de computación.

“Muchos sistemas heredados continúan siendo un apoyo para las funciones básicas del negocio y son “indispensables” para este” Liu (1998).

CARACTERÍSTICAS DEL SOFTWARE HEREDADO

Se caracteriza por su LONGEVIDAD e IMPORTANCIA CRÍTICA PARA EL NEGOCIO, desafortunadamente, otra característica presente es la MALA CALIDAD, tienen diseños que no son susceptibles de extenderse, CÓDIGO CONFUSO, DOCUMENTACIÓN MALA O INEXISTENTE, CASOS Y RESULTADOS DE PRUEBAS que nunca se ARCHIVARON, una historia de los cambios mal administrada… la lista es muy larga.

CARACTERÍSTICAS
- Longevidad
- importancia crítica para el negocio
- Mala calidad (código confuso, doc. mala…)

A pesar de esto dan APOYO A LAS FUNCIONES BÁSICAS DEL NEGOCIO Y SON INDIPENSABLES PARA ESTE. Entonces ¿Qué hacer? La respuesta es no hacer nada hasta que el sistema heredado tenga un cambio significativo, si el software heredado satisface las necesidades del usuario y corre de manera confiable, entonces no falla ni necesita repararse, sin embargo, conforme pase el tiempo será frecuente que los SISTEMAS DE SOFTWARE EVOLUCIONEN POR UNA O VARIAS de las siguientes razones:

- El software debe adaptarse para que cumpla las NECESIDADES DE LOS NUEVOS AMBIENTES del cómputo y la tecnología.

- El software DEBE SER MEJORADO para implementar NUEVOS REQUERIMIENTOS del negocio.
- El software debe AMPLIARSE para que sea OPERABLE con otros SISTEMAS o BASES DE DATOS MODERNOS.
- La arquitectura del software debe REDISEÑARSE PARA HACERLA VIABLE DENTRO DE UN AMBIENTE DE REDES.

MITOS DEL SOFTWARE

Creencias acerca del Software y de los procesos empleados para construirlos.

• MITOS DE LA ADMINISTRACIÓN

Administradores con responsabilidades sobre el Software, están bajo presión por MANTENER EL PRESUPUESTO, EVITAR que los ITINERARIOS se EXTIENDAN y MEJORAR LA CALIDAD.

MITO 1

Ya se tiene un libro lleno de estándares y procedimientos para la construcción de Software y ¿Esto proporcionará a mi gente todo lo necesario?

REALIDAD 1

No porque exista se sepa, se use, sea actual, adaptable, etc.

MITO 2

Si se está atrasado es posible CONTRATAR más PROGRAMADORES para terminar a tiempo (concepto de la HORDA MONGOLA)

REALIDAD 2

“Agregar gente a un proyecto de Software atrasado lo atrasa más” BROOKS, porque se debe invertir tiempo en la enseñanza a los recién llegados. Se puede agregar gente, pero solo de una manera planeada y bien coordinada.

MITO 3

Si decido subcontratar el proyecto de Software a un tercero, puedo relajarme y que esa compañía lo construya.

REALIDAD 3

Si una organización no entiende cómo administrar y controlar internamente los proyectos de Software, de manera invariable entrará en conflicto al subcontratar este tipo de proyecto.

• MITOS DEL CLIENTE

Puede ser la persona del escritorio del al lado, un grupo de técnicos del piso de abajo, algún departamento de la empresa, los clientes creen en mitos por desinformación, los mitos conducen a expectativas falsas (del cliente) y a insatisfacción con el desarrollador.

MITO 1

Un enunciado general de los objetivos es suficiente para comenzar a escribir programas, los detalles se pueden afinar después.

REALIDAD 1

Un enunciado ambiguo de los objetivos es la receta perfecta para el desastre. Los requerimientos precisos se logran mediante comunicación continua y efectiva entre el cliente y desarrollador.

MITO 2

Los requerimientos del proyecto cambian de manera continua pero el cambio puede ajustarse con facilidad porque el software es flexible.

REALIDAD 2

Si existen cambios, PERO EL IMPACTO del cambio varía de acuerdo CON EL MOMENTO en que este se introduce. Cambios en etapas tempranas, antes del código o diseño el impacto del costo es pequeño, conforme pasa el tiempo el IMPACTO en el costo CRECE.

MITOS DEL DESARROLLADOR

MITO 1

Una vez que el programa ha sido escrito y puesto a funcionar, el trabajo está terminado.

REALIDAD 1

La industria indica que entre el 60% y 80% de todo el esfuerzo en el software se realizará después de que el sistema haya sido entregado al cliente por primera vez.

MITO 2

Mientras el programa no se esté ejecutando, no existe forma de evaluar su calidad.

REALIDAD 2

Los mecanismos para asegurar la calidad se pueden aplicar desde el inicio de un proyecto: revisión técnica formal, revisiones del software, son un “filtro de calidad”.

MITO 3

El único producto del trabajo que puede entregarse para tener un proyecto exitoso es el programa en funcionamiento.

REALIDAD 3

Existe también la documentación que proporciona un fundamento para la ingeniería exitosa y representa una guía para el mantenimiento de software.

MITO 4

La ingeniería de software obligará a emprender la creación de una documentación voluminosa e innecesaria y de manera invariable tornará más lento el proceso.

REALIDAD 4

La ingeniería de software no se refiere a la elaboración de documentos, está relacionada con la creación de la calidad y por ende conduce a la reducción de trabajos redundantes y por ende menor tiempo de entrega.

REALIDADES PARA USAR INGENIERÍA DE SOFTWARE

- **Entender el problema antes de desarrollar una aplicación de software. -** Deben escucharse muchas opiniones y hacer un esfuerzo para entender el problema.
- **El diseño se ha vuelto una actividad crucial. -** En la actualidad grandes equipos de personas crean programas de cómputo que antes eran elaborados por un solo individuo, hoy en día la complejidad de los sistemas y productos basados en computadoras demandan atención

cuidadosa de las interacciones de todos los elementos del sistema que el diseño se ha vuelto una actividad crucial.

- **El Software debe tener alta calidad.** - Muchas organizaciones sociales, empresariales, gubernamentales e individuos depende del software para tomar decisiones estratégicas y tácticas, si el software falla pueden experimentar fallas catastróficas es de vital importancia que el software tenga alta calidad.
- **Facilidad para Recibir Mantenimiento.-** Las demandas para adaptar y mejorar una aplicación de acuerdo a su longevidad y base de usuarios a medida que va pasando el tiempo crece la probabilidad de recibir mejoras o actualización es por esto que el software debe tener facilidad para recibir mantenimiento.

DEFINICIONES DE INGENIERÍA DE SOFTWARE

La Ingeniería de Software es el establecimiento y USO DE PRINCIPIOS fundamentales de la ingeniería, tiene como objeto desarrollar en forma ECONÓMICA software que sea CONFIABLE y que trabaje con EFICIENCIA en MÁQUINAS REALES.

IS
- Desarrollar en forma económica
- Confiable
- Eficiente
- En máquinas reales

La Ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.

- ### Fundamento en el que se apoya la IS:

Compromiso con la CALIDAD

- ### Capas de la Ingeniería de Software:

PROCESO: Define una estructura que debe establecerse para la obtención eficaz de tecnología de ingeniería de software. El proceso de Software forma la base para el control de la administración de proyectos de software y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos, etc.) Se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada.

MÉTODOS: Proporcionan la experiencia técnica para elaborar software. Incluyen un conjunto amplio de tareas como comunicación, análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo. Los métodos se basan en un conjunto de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas.

HERRAMIENTAS: Proporcionan un apoyo automatizado o semi-automatizado para el proceso y los métodos. Cuando se integran las herramientas de modo que la información creada por una pueda ser utilizada por otra, queda establecido un sistema llamado ingeniería de software asistido por computadora que apoya el desarrollo del software.

EL PROCESO DE SOFTWARE

Un proceso es un conjunto de **actividades, acciones y tareas** que se ejecutan cuando va a crearse algún producto del trabajo.

- Una **actividad** busca lograr un objetivo amplio (por ejemplo, comunicación con los participantes).
- Una **acción** (diseño de la arquitectura) es un conjunto de tareas que producen un producto importante del trabajo (por ejemplo, un modelo del diseño de la arquitectura).
- Una **tarea** se centra en un objetivo pequeño, pero bien definido (por ejemplo, realizar una prueba unitaria) que produce un resultado tangible.

Un proceso no es una prescripción rígida de cómo elaborar un software, por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo.

ACTIVIDADES ESTRUCTURALES DEL PROCESO

La Estructura del Proceso consta de 5 actividades:

1. COMUNICACIÓN: Antes de que comience cualquier trabajo técnico tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes respecto del proyecto y reunir los requerimientos que ayuden a definir las características y funciones del software.
2. PLANEACIÓN: La actividad de planeación crea un mapa que guía al equipo, este mapa es llamado *plan del proyecto del software,* define el trabajo de ingeniería de software al describir las **tareas técnicas** para realizar los **riesgos** probables, los **recursos** que se requieren, los **productos del trabajo** que se obtendrán y una **programación de las actividades**.
3. MODELADO: Un diseñador, arquitecto, carpintero, etc. crea un bosquejo del objeto por hacer a fin de entender el panorama general, cómo se verá arquitectónicamente, cómo ajustan entre si las partes constituyentes y muchas características más, un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.
4. CONSTRUCCIÓN: Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.
5. DESPLIEGUE: El software se entrega al consumidor que lo evalúa y le da retroalimentación, misma que se basa en dicha evaluación.

ACTIVIDADES SOMBRILLA

Las actividades estructurales del proceso son complementadas con por cierto número de ACTIVIDADES SOMBRILLA.

1. SEGUIMIENTO Y CONTROL DE DEL PROYECTO DE SOFTWARE: Permite que el equipo de software evalúe el progreso comparándolo con el plan del proyecto y tome cualquier acción necesaria para apegarse a la programación de actividades. (COMPARACIÓN)
2. ADMINISTRACIÓN DE RIESGOS: Evalúa los riesgos que puedan afectar el resultado del proyecto o la calidad del producto.
3. ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE: Define y ejecuta las actividades requeridas para garantizar la calidad del software.
4. REVISIONES TÉCNICAS: Evalúa los productos del trabajo de la ingeniería de software a fin de descubrir y eliminar errores antes de que se propaguen a la siguiente actividad.
5. MEDICIÓN: Define y reúne mediciones del proceso, proyecto y producto para ayudar al equipo a entregar el software que satisfaga las necesidades de los participantes; puede usarse junto con todas las demás actividades estructurales y sombrillas.
6. ADMINISTRACIÓN DE LA CONFIGURACIÓN DEL SOFTWARE: Administra los efectos del cambio a lo largo del proceso del software.
7. ADMINISTRACIÓN DE LA REUTILIZACIÓN: Define criterios para volver a usar el producto del trabajo (incluso los componentes del software) y establece mecanismos para obtener componentes reutilizables.
8. PREPARACIÓN Y PRODUCCIÓN DEL PRODUCTO DEL TRABAJO: Agrupa las actividades requeridas para crear productos del trabajo, tales como modelos, documentos, registros, formatos y listas.

LA PRÁCTICA DE LA INGENIERÍA DE SOFTWARE

LA ESENCIA DE LA PRÁCTICA:

1. Entender el Problema (comunicación y análisis)
2. Planear la solución (modelado y diseño del software)
3. Ejecutar el Plan (generación del código)
4. Examinar la exactitud del resultado (probar y asegurar la calidad)

Estas etapas conducen a una serie de preguntas esenciales:

ENTENDER EL PROBLEMA

- ¿Quiénes tienen que ver con la solución del Problema? Es decir ¿Quiénes son los participantes?
- ¿Cuáles son las incógnitas? ¿Cuáles datos, funciones y características se requieren para resolver el problema en forma apropiada?
- ¿Puede fraccionarse el problema? ¿Es posible representarlo con problemas más pequeños que sean más fáciles de entender?
- ¿Es posible representar gráficamente el problema? ¿Puede crearse un modelo de análisis?

PLANEAR LA SOLUCIÓN

Antes de escribir el código haga un pequeño diseño.

- ¿Has vito antes problemas similares? ¿Hay patrones reconocibles en una solución potencial? ¿Hay algún software existente que implemente los datos, funciones y características que se requieren?

- ¿Has resuelto un problema similar? Sí, es así ¿son reutilizables los elementos de la solución?
- ¿Pueden definirse problemas más pequeños? Sí así fuera, ¿Hay soluciones evidentes para éstos?
- ¿Es capaz de representar una solución en una forma que lleve a su implementación eficaz? ¿Es posible crear un modelo de diseño?

EJECUTAR EL PLAN

Al querer tratar de realizar el plan diseñado puede haber desviaciones inesperadas y es posible que descubra un camino mejor a medida que avanza, pero el plan le permitirá avanzar sin que se pierda.

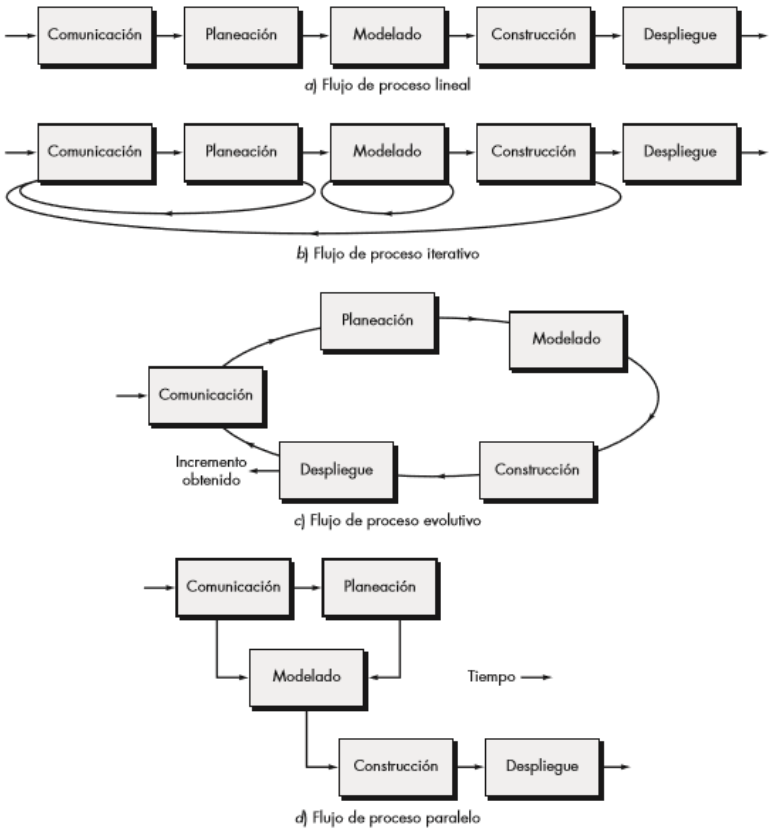
- ¿Se ajusta la solución al plan? ¿El código fuente puede apegarse al modelo del diseño?
- ¿Es probable que cada parte componente de la solución sea correcta? ¿El diseño y el código se han revisado o mejor aún, se han hecho pruebas respecto de la corrección del algoritmo?

EXAMINAR EL RESULTADO

No se puede estar seguro de que la solución sea perfecta, pero sí de que se ha diseñado un número suficiente de pruebas para descubrir tantos errores como sea posible.

- ¿Puede probarse cada parte componente de la solución? ¿Se ha implementado una estrategia razonable para hacer pruebas?
- ¿La solución produce resultados que se apegan a los datos, funciones y características que se requieren? ¿El software se ha validado contra todos los requerimientos de los participantes?

FLUJO DE PROCESO



PROYECTO DE SOFTWARE PEQUEÑO SOLICITADO POR UNA PERSONA EN UNA UBICACIÓN REMOTA.

Con requerimientos sencillos y directos:

- Hacer contacto con el participante por vía telefónica
- Analizar los requerimientos y tomar notas
- Organizar las notas por escrito en una formulación breve de los requerimientos.
- Enviar correo electrónico al participante para que revise y apruebe.

Si el proyecto fuera más complejo, con muchos participantes y cada uno con un distinto conjunto de requerimientos, la actividad de comunicación puede tener 6 acciones distintas:

1. Concepción
2. Indagación
3. Elaboración
4. Negociación
5. Especificación
6. Validación

MODELOS DE PROCESOS PRESCRIPTIVO

Los modelos de proceso prescriptivo fueron propuestos originalmente para poner orden en el caos del desarrollo de software.

Todos los modelos del proceso del software pueden incluir las 5 actividades estructurales, pero cada una pone distinto énfasis en ellas y define en forma diferente el flujo de proceso que invoca cada actividad estructural.

MODELO CASCADA: Nació en 1970 (Winston W. Royce) llamado ciclo de vida clásico también Lineal Secuencial antes de codificar debemos diseñar el software, además probarlo antes de construirlos y ponerlo en operación.

Ventajas

1. Modelo y planificación fácil y sencillo
2. Sus fases son conocidos por los desarrolladores
3. Los usuarios lo pueden comprender fácilmente
4. Ayuda a detectar errores en las primeras etapas a bajo costo

Desventajas

1. Alto Riesgo en Sistemas nuevos debido a problemas en las especificaciones y en el diseño
2. Es difícil que el cliente exponga explícitamente todos los requisitos al principio
3. El cliente debe tener paciencia pues obtendrá el producto al final del ciclo de vida.

Es raro que los proyectos reales sigan el flujo secuencial propuesto por el modelo. Aunque el modelo lineal acepta repeticiones, lo hace en forma indirecta. Como resultado, los cambios generan confusión conforme el equipo del proyecto avanza.

En un análisis interesante de proyectos reales, Bradac encontró que la naturaleza lineal del ciclo clásico llega a “estados de bloqueo” en los que ciertos miembros del equipo del proyecto deben esperar a otros a fin de terminar tareas interdependientes. Los estados de bloqueo tienden a ocurrir más al principio y al final de un proceso lineal secuencial.

MODELO EVOLUTIVO

Permite desarrollar de manera iterativa, nuevas versiones de Software cada vez más completas.

- MODELO INCREMENTAL
- MODELO EN ESPIRAL
- MODELO EN ESPIRAL VICTORIA – VICTORIA (WIN-WIN)
- MODELO DE DESARROLLO CONCURRENTE.

DESARROLLO EVOLUTIVO

- *DESARROLLO EXPLORATORIO:* Explora los requerimientos del cliente, el sistema evoluciona agregando nuevos atributos propuestos por el cliente.
- *DESARROLLO DE PROTOTIPOS:* Comprende los requerimientos del cliente y entonces desarrolla una definición mejorada de los requerimientos para el sistema.

MODELO INCREMENTAL

Fue propuesto por Harlan Mills en 1980, para reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Este modelo se conoce también bajo las siguientes denominaciones:

- Método de las comparaciones limitadas sucesivas.
- Ciencia de salir del paso.
- Método de atacar el problema por ramas.

El Modelo Incremental combina elementos del Modelo Lineal Secuencial con la filosofía interactiva de Construcción de Prototipos. Como se muestra en la Figura 1, el modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. El primer incremento generalmente es un producto esencial denominado núcleo.

CARACTERÍSTICAS:

- Se evitan proyectos largos y se entrega "algo de valor" a los usuarios con cierta frecuencia.
- El usuario se involucra más.
- Difícil de evaluar el costo total.
- Difícil de aplicar a los sistemas transaccionales que tienden a ser integrados y a operar como un todo.
- Requiere gestores experimentados.
- Los errores en los requisitos se detectan tarde.
- El resultado puede ser positivo.

VENTAJAS:

- Con un paradigma incremental se reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial.

- También provee un impacto ventajoso frente al cliente, que es la entrega temprana de partes operativas del software.
- El modelo proporciona todas las ventajas del modelo en Cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento.
- Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos.

DESVENTAJAS:

- El modelo incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido y/o de alto índice de riesgos.
- Requiere de mucha planeación, tanto administrativa como técnica.
- Requiere de metas claras para conocer el estado del proyecto.

CONCLUSIÓN:

Un modelo incremental lleva a pensar en un desarrollo modular, con entregas parciales del producto Software denominados "incrementos" del sistema, que son escogidos en base a prioridades predefinidas de algún modo.

El modelo permite una implementación con refinamientos sucesivos (ampliación y/o mejoras). Con cada incremento se agrega nueva funcionalidad o se cubren nuevos requisitos o bien se mejora la versión previamente implementada del producto software.

MODELO EN ESPIRAL

Modelo de Desarrollo evolutivo propuesto por Barry Boehm en 1988, utiliza prototipos como apoyo. La forma de espiral representa una iteración (repetición) de procesos que a medida que se van entregando los prototipos y éstos son revisados por los clientes.

Las regiones de tareas que componen este modelo son:

- **Comunicación con el cliente:** las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.
- **Planificación:** las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto. Son todos los requerimientos.
- **Análisis de riesgos:** las tareas requeridas para evaluar riesgos técnicos y otras informaciones relacionadas con el proyecto.
- **Ingeniería:** las tareas requeridas para construir una o más representaciones de la aplicación.
- **Construcción y adaptación:** las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario.
- **Evaluación del cliente:** las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementación durante la etapa de instalación.

El movimiento de la espiral, ampliando con cada iteración su amplitud radial, indica que cada vez se van construyendo versiones sucesivas del software, cada vez más completas.

Uno de los puntos más interesantes del modelo, es la introducción al proceso de desarrollo a las actividades de análisis de los riesgos asociados al desarrollo y a la evaluación por parte del cliente de los resultados del software.

MODELO ESPIRAL WIN WIN

Barry Boehm, años después realizó una variante o actualización del ciclo de vida en espiral que definió en 1988.

Con la variante trata de ajustarse más a la realidad de lo que es un proyecto de desarrollo de software, en el cual el resultado final no es exactamente la implementación del catálogo de requisitos, sino que una vez definido se renegocia el alcance del mismo, incluso en diversas partes del proyecto, entre cliente y proveedor con el objetivo de intentar que ambas partes queden satisfechas (aunque en la mayoría de los casos, cada parte solo se mire su ombligo).

La variante se basa en la inclusión de tres etapas o regiones al principio:

1.- Identificar las partes interesadas (stakeholders) para esta nueva iteración del producto: Es necesario definir los interlocutores que serán de áreas que se verán afectadas por el resultado final de la nueva versión. Estos interlocutores serán del área del cliente (puede haber más de uno) y del proveedor.

2.- Identificar las condiciones de victoria de las partes interesadas en el proyecto: Se concreta cuáles son las condiciones que requiere cada parte para que se sienta satisfecha una vez realizada esta versión.

3a.- Reunir las condiciones de victoria: Con las etapas anteriores se han definido unos objetivos generales para la versión y se obtiene conocimiento de los objetivos particulares de cada parte. Ahora toca negociar hasta dónde realmente se va a llegar y cómo, intentando llegar a una solución en la que todos ganen (cliente y proveedor).

Las siguientes etapas tienen una correspondencia con algunas variantes, con la versión inicial del ciclo de vida en espiral:

3b.- Establecer los objetivos, restricciones y alternativas del siguiente nivel: Teniendo en cuenta los objetivos y acuerdos de las fases anteriores, se definirían los requisitos de esta versión, además de especificarse diversas alternativas en el caso de que existan.

4.- Evaluar las alternativas del producto y del proceso y resolución de riesgos: Se realizaría el análisis del riesgo típico de los modelos en espiral.

5.- Definir el siguiente nivel del producto y del proceso, incluyendo particiones: Esta etapa completaría el proceso de análisis y realizaría el diseño y la construcción.

6.- Validar las definiciones del producto y del proceso: Comprendería la implantación y aceptación de la versión, verificándose si el resultado de la iteración satisface el alcance indicado.

7.- Revisión y comentarios: Tocaría hacer inventario, medir el nivel de satisfacción de las partes, el nivel de cumplimiento de objetivos con el objetivo sobre todo de intentar aprender de los errores para mejorar en versiones sucesivas y de detectar correcciones y mejoras a realizar en el producto.

Creo que lo más interesante del modelo es que se especifique de forma explícita la necesidad de que las partes negocien para llegar a un acuerdo satisfactorio para todos, por eso esta variante recibe el nombre de Win Win (ganar-ganar). Aunque es complicado alcanzar un equilibrio en el que ambas partes ganen a un 50%, sí que es fundamental que independientemente de si uno es un poco más ganador que el otro, todas las partes estén convencidas en que el acuerdo es bueno.

En cualquier caso sigue sin ser absolutamente realista con respecto al transcurso normal de un proyecto de desarrollo de software, donde la negociación se extiende en muchos proyectos hasta el mismo proceso de construcción del sistema de información, no obstante, si los incrementos no son muy grandes no tendría por qué extenderse tanto la negociación, no obstante, como en este tipo de modelos de ciclo de vida, en cada iteración (incluida la primera) se intenta tener un producto funcionando, salvo que éste sea trivial, las primera etapa por lo menos tendrá tamaño suficiente para que en muchos casos nos encontremos con casos donde se estén negociando aspectos del proyecto hasta el final.

5. Metodología

Estrategias de enseñanza-aprendizaje propuestas

La asignatura se desarrolla bajo un enfoque de **aprendizaje basado en problemas y proyectos**, fomentando el pensamiento crítico y la aplicación práctica del conocimiento.

Entre las estrategias utilizadas se incluyen:

- **Aprendizaje basado en proyectos (ABP):** Los estudiantes trabajan en proyectos reales o simulados, aplicando los conocimientos adquiridos.
- **Trabajo en equipo:** Se fomenta la colaboración en grupos para resolver problemas de software de manera efectiva.
- **Estudio de casos:** Se analizan casos reales de éxito y fracaso en la industria del software para extraer lecciones prácticas.
- **Uso de herramientas tecnológicas:** Se incorporan plataformas como GitHub, Trello y herramientas de desarrollo para la gestión de proyectos.

Actividades para reforzar el aprendizaje

- Desarrollo de un proyecto de software en equipo**, aplicando metodologías ágiles.**
- Análisis y discusión de artículos científicos y casos de estudio.
- Prácticas de programación y modelado de software.
- Autoevaluaciones y coevaluaciones para mejorar la comprensión y el trabajo en equipo.

6. Evaluación

Criterios y métodos de evaluación

La evaluación se basará en la participación activa, el desarrollo de proyectos y la aplicación de conocimientos. Se utilizarán los siguientes criterios:

- **Proyectos prácticos:** 20%
- **Exámenes teóricos y prácticos:** 30%
- **Trabajo en equipo y participación:** 40%
- **Autoevaluación y coevaluación:** 10%

Pautas para autoevaluación y coevaluación

- Los estudiantes reflexionarán sobre su desempeño en proyectos y actividades.
- Se implementarán rúbricas de evaluación para valorar el trabajo en equipo y el cumplimiento de objetivos.
- Se fomentará la retroalimentación constructiva entre compañeros.

7. Bibliografía

Fuentes y referencias utilizadas y recomendadas

- Pressman, R. S. (2020). *Ingeniería de Software: Un enfoque práctico*. McGraw-Hill.
- UNEMI (NOV, 2019). Ingeniería de Software I: Modelos de Proceso de Software. https://sga.unemi.edu.ec/media/recursotema/Documento_202169165012.pdf

8. Anexos (opcional)

Ejercicios prácticos

ORDEN	TEMAS	EJERCICIO PRÁCTICO
01	DOMINIOS DE APLICACIÓN DEL SOFTWARE	Pedir que los estudiantes expongan sobre las diferentes aplicaciones de software que existen hoy en día en cada una de las categorías: Sistemas, Aplicación a medida, ingeniería y ciencia, incrustado, línea de producto, app web y de inteligencia artificial
02	CONVERSIÓN DE INFORMACIÓN: MIGRACIÓN, FUSIÓN, SOFTWARE HEREDADO	Realizar una comparación alegórica sobre la migración y recalcar las razones del cambio, pero también lo que no puede cambiar, plantear fusiones de estilos de comida para relacionar el concepto de fusión y pedir características de una herencia para calificar las características que conlleva de tal manera que los estudiantes puedan relacionar los conceptos con el software.
03	MITOS Y REALIDADES DEL SOFTWARE: MITOS DEL CLIENTE	En los mitos del cliente tomar un alumno como cliente y darle todas las indicaciones que el necesita para un software, pero pedirle que sea lo más hermético posible con sus compañeros, de tal manera que los estudiantes puedan constatar todos los mitos del cliente.
04	MITOS Y REALIDADES DEL SOFTWARE: MITOS DEL ADMINISTRADOR	Tomar un alumno como el administrador de un proyecto y pedirles a sus compañeros que se ingenien la cantidad de peticiones y problemas a los que tendrá que solucionar, especialmente el de la horda mongola, realizar el cambio de estudiantes para realizar el mismo ejercicio varias veces.
05	MITOS Y REALIDADES DEL SOFTWARE: MITOS DEL DESARROLLADOR	En cuanto al mito del desarrollador pedirle a los estudiantes que evalúen un sistema simulacro y que

		pidan que cambios podrían tener, desde la interface hasta su funcionamiento, de tal manera que constaten que entre el 60% y 80% de todo el esfuerzo en el software se realizará después de que el sistema haya sido entregado al cliente por primera vez.
06	CAPAS DE LA INGENIERÍA DE SOFTWARE: PROCESO, MÉTODOS Y HERRAMIENTAS	Plantear a los estudiantes la creación de un emprendimiento y relacionarlo con estas tres capas de la ingeniería del software, como se va dando el proceso de creación que conlleva la planeación, los métodos más prácticos y seguros que nos den el camino más óptimo y las herramientas o recurso material o físico para conseguirlo.
07	ACTIVIDADES ESTRUCTURALES DEL SOFTWARE: COMUNICACIÓN	Poner un alumno como cliente frente al grupo de desarrolladores, hacerlo por equipo y poder establecer las bases de una buena comunicación y sobre todo el entendimiento del problema que solucionará con el software.
08	ACTIVIDADES ESTRUCTURALES DEL SOFTWARE: PLANEACIÓN	En la planeación realizar un proyecto ficticio en donde los alumnos puedan plasmar cada uno de los puntos de la planeación y exponerlos con ese ejemplo práctico, tales como distribución de las tareas técnicas, realización de un cronograma de actividades, evaluar los riesgos y realizar su plan de contingencia, realizar un presupuesto y que software van a realizar.
09	FLUJO DEL PROCESO DEL SOFTWARE	Una vez que se han visto todos los modelados, identificar que modelado le quedaría perfecto a un tipo de proyecto, realizar una lluvia de ideas con los alumnos.

Elaborado Por:

_____

Ing. Andrea Cartuche J.
Docente