Ensayo

Nombre del alumno: Yahir Aguilar Sicalhua.
Nombre del tema: Fundamentos de la Ingeniería del Software e Ingeniería de Requisitos.
Parcial: 1.
Nombre de la materia: Ingeniería en Software.
Nombre del profesor: Juan José Ojeda Trujillo.
Nombre de la licenciatura: Ingeniería en Sistemas Computacionales.
Cuatrimestre: 8.

En el vertiginoso mundo de la tecnología, la Ingeniería del Software emerge como una disciplina esencial, sirviendo como cimiento para el desarrollo eficiente y sostenible de aplicaciones y sistemas informáticos. Este ensayo explorará los fundamentos que subyacen en esta rama ingenieril, desentrañando los principios clave que guían la creación de software robusto y de alto rendimiento. Desde la conceptualización de requisitos hasta la implementación y mantenimiento, la ingeniería del software se erige como el arte y la ciencia de traducir ideas en soluciones digitales, impulsando la innovación y transformando la manera en que interactuamos con la tecnología. En este análisis, profundizaremos en los pilares fundamentales que sostienen esta disciplina, destacando su relevancia en la era digital actual.

FUNDAMENTOS DE LA INGENIERÍA DEL SOFTWARE.

1.1. DEFINICIÓN Y OBJETIVOS DE LA INGENIERÍA DEL SOFTWARE.

Ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo operación (funcionamiento) y mantenimiento del software: es decir, la aplicación de ingeniería al software.

El concepto de ingeniería del software surgió en 1968, tras una conferencia en Garmisch (Alemania) que tuvo como objetivo resolver los problemas de la crisis del software. Este fue ocasionado por el avance desmesurado del hardware lo que hacía el software cada vez más completo y generalmente nunca se terminaba a tiempo.

1.2. CARACTERISITICAS Y APLICACIONES DEL SOFTWARE.

Un producto software puede ser juzgado según lo que ofrece y la manera en que se puede usar. El software debe satisfacer en los siguientes aspectos:

- Operacional
- Transicional
- Mantenimiento Operacional

Esto nos cuenta lo bien que funciona el software en operaciones. Se puede medir en base a:

- Presupuesto
- Usabilidad
- Eficiencia
- Exactitud
- Funcionalidad
- Dependabilidad
- Seguridad informática
- Seguridad

1.3. EVOLUCIÓN HISTÓRICA DEL SOFTWARE.

El proceso de desarrollo de un producto software usando principios y métodos de Ingeniería de software, se denomina Evolución del Software. Esto incluye el desarrollo inicial del software, mantenimiento y actualizaciones, hasta que el producto deseado finalmente es desarrollado, lo que satisface los requerimientos esperados.

La evolución empieza con un proceso de recogida de requisitos. Luego los desarrolladores crean un prototipo inicial del software y se muestra a los consumidores para tener un feedback en una etapa temprana del desarrollo del producto de software. Los consumidores sugieren cambios, los cuales irán mejorando con actualizaciones y tareas de mantenimiento de manera progresiva. Este proceso cambia el software original hasta llegar al producto deseado.

Incluso después de que el consumidor tenga el software en sus manos, el avance de la tecnología y los cambios de requisitos fuerzan al producto software a cambiar en acorde a estos. Volver a cerrar software desde cero, e ir cumpliendo uno por uno los requisitos no son viable. La única solución viable y económica es actualizar el software ya existente para que se adecue satisfactoriamente con los requisitos más recientes.

1.4. LEYES DE EVOLUCIÓN DEL SOFTWARE.

Lehman formuló leyes para la evolución del software. Dividió el software en 3 categorías distintas:

'S-type' ('static-type', tipo estático): Es un tipo de software, que funciona estrictamente según se ha definido especificaciones y soluciones. La solución y el método mediante el cual se consigue, se deben entender de inmediato antes de empezar a codificar. El software 's- type' está menos sujeto a cambios, de ahí que sea el más simple de todos. Por ejemplo, el programa de calculadora, para computación matemática. 'P-type' ('practical-type', tipo práctico): Este es un software con una colección de procedimientos. Esto se define exactamente por lo que pueden hacer los procedimientos. En este software, las especificaciones se pueden describir, pero la solución no es obvia al instante. Por ejemplo, software de juegos. 'E-type' ('embedded-type', tipo embebido o empotrado): Este software funciona estrechamente como requisito del entorno del mundo real. Este software tiene un alto grado de evolución ya que hay varios cambios en las leyes, impuestos, etc. en las situaciones del mundo real. Por ejemplo, el software de comercio en línea.

1.5. PARADIGMAS DE SOFTWARE.

Los paradigmas de Software son métodos y pasos, que se llevan a cabo mientras el software se diseña. Hay muchos métodos que se han propuesto y que funcionan hoy en día, pero necesitamos ver donde se ubican estos paradigmas en el marco de la Ingeniería de software. Estos se pueden combinar en varias categorías, en las que cada uno de ellos contiene a la otra:

El paradigma de programación es una parte del paradigma de diseño de Software y más adelante también se considera parte del paradigma de desarrollo de Software.

1.6. PERSPECTIVA GENERAL DE LA INGENIERIA DEL SOFTWARE.

Inicialmente la programación de las computadoras era un arte que no disponía de métodos sistemáticos en los que poder basarse para la realización de productos software. Se realizaban sin ninguna planificación.
Posteriormente, desde mediados de los 60 hasta finales de los 70 se caracterizó por el establecimiento del software como un producto que se desarrollaba para una distribución general. En esta época nació lo que se conoce como el mantenimiento del software que se da cuando cambian los requisitos de los usuarios y se hace necesaria la modificación del software. El esfuerzo requerido para este mantenimiento era en la mayoría de los casos tan elevado que se hacía imposible su mantenimiento.

1.7. PROCESOS, MÉTODOS Y HERRAMIENTAS.

Un método de ingeniería del software es un enfoque estructurado para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable. Métodos como Análisis Estructurado (DeMarco, 1978) y JSD (Jack- son, 1983) fueron los primeros desarrolladores en los años 70. Estos métodos intentaron identificar los componentes funcionales básicos de un sistema de tal forma que los métodos orientados a funciones aún se utilizan ampliamente. En los años 80 y 90, estos métodos orientados a funciones fueron complementados por métodos orientados a objetos, como los propuestos por Booh (1994) y Rumbaugh et al., 1991). Estos diferentes enfoques se han integrado a un solo enfoque unificado basado en UML (Lenguaje de Modelado Unificado).

1.8. MODELO CLÁSICO O LINEAL, MODELO EN CASCADA.

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

1.9. PARADIGMA DE DESARROLLO DE SOFTWARE.

El Paradigma de desarrollo de Software ayuda al desarrollador a escoger una estrategia para desarrollar el software. El paradigma de desarrollo software tiene su propio set de herramientas, métodos y procedimientos, los cuales son expresados de forma clara, y define el ciclo de vida del desarrollo del software.

1.10. CONSTRUCCIÓN DE PROTOTIPO.

El modelo de prototipos permite que todo el sistema, o algunos de sus partes, se construyan rápidamente para comprender con facilidad y aclarar ciertos aspectos en los que se aseguren que el desarrollador, el usuario, el cliente estén de acuerdo en lo que se necesita así como

también la solución que se propone para dicha necesidad y de esta forma minimizar el riesgo y la incertidumbre en el desarrollo, este modelo se encarga del desarrollo de diseños para que estos sean analizados y prescindir de ellos a medida que se adhieran nuevas especificaciones, es ideal para medir el alcance del producto, pero no se asegura su uso real.

1.11. CICLO DE VIDA DE UN SISTEMA BASADO EN PROTOTIPO.

Una maqueta o prototipo de pantallas muestra la interfaz de la aplicación, su cara externa, pero dicha interfaz está fija, estática, no procesa datos. El prototipo no tiene desarrollada una lógica interna, sólo muestra las pantallas por las que irá pasando la futura aplicación.
Por su parte, el prototipo funcional evolutivo desarrolla un comportamiento que satisface los requisitos y necesidades que se han entendido claramente. Realiza, por tanto, un proceso real de datos, para contrastarlo con el usuario. Se va modificando y desarrollando sobre la marcha, según las apreciaciones del cliente. Esto ralentiza el proceso de desarrollo y disminuye la fiabilidad, puesto que el software está constantemente variando, pero, a la larga, genera un producto más seguro, en cuanto a la satisfacción de las necesidades del cliente.

1.12. MODELOS EVOLUTIVOS.

Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez más completas del software.
El software evoluciona con el tiempo. Los requisitos del usuario y del producto suelen cambiar conforme se desarrolla el mismo. Las fechas de mercado y la competencia hacen que no sea posible esperar a poner en el mercado un producto absolutamente completo, por lo que se aconsejable introducir una versión funcional limitada de alguna forma para aliviar las presiones competitivas.

INGENIERÍA DE REQUISITOS.

2.1. ANÁLISIS DE REQUERIMIENTOS.

Los requerimientos permiten que los desarrolladores expliquen cómo han entendido lo que el cliente pretende del sistema. También, indican a los diseñadores qué funcionalidad y qué características va a tener el sistema resultante. Y, además, indican al equipo de pruebas qué demostraciones llevar a cabo para convencer al cliente de que el sistema que se le entrega es lo que solicitó. Las características de los requerimientos mencionados en el estándar IEEE830 los explica [Pfleeger, 2002]

2.2. TIPOS DE REQUERIMIENTOS.

Según el estándar internacional de Especificación de Requerimientos IEEE830, los documentos de definición y especificación de requerimientos deben contemplar los siguientes aspectos resumidos por [Pfleeger, 2002] como se indica a continuación:

Ambiente físico
Interfaces
Usuarios y factores humanos
Funcionalidad
Documentación
Datos
Recursos
Seguridad

2.3. IDENTIFICACIÓN, ANÁLISIS, NEGOCIACIÓN.

Métodos generales de entrevistas.

La entrevista es una forma de recoger información de otra persona a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada, [Braude, 2003] distingue las siguientes fases:

- Preparación: El entrevistador debe documentarse e investigar la situación de la organización, analizando los documentos de la empresa disponible. Hay que intentar minimizar el número de entrevistados, hay que considerar las entrevistas de cortesía, analizar el perfil de los entrevistados, definir el objetivo y el contenido de la entrevista, planificar el lugar y la hora en la que se va a desarrollar la entrevista es conveniente realizarla en un lugar confortable.

2.4. VALIDACIÓN Y GESTIÓN DE REQUISITOS.

Es un proceso que consta de cuatro pasos:

1. Estudio de viabilidad
2. Recogida de requisitos
3. Requisitos del Software
4. Validación de los requisitos de Software

2.5. VALIDACIÓN DE LOS REQUISITOS DE SOFTWARE.

Después del desarrollo de los requisitos, los que se mencionen en este documento serán validados. El usuario puede que pida soluciones ilegales y poco prácticas, y los expertos puede que interpreten los requisitos de forma incorrecta. Estos resultados se incrementan en coste si no se cortan de raíz. Los requisitos se pueden evaluar en contraste con las siguientes condiciones:

- Si pueden ser implementados de manera práctica.
- Si son válidos a nivel de funcionalidad y dominio del software

- Si hay alguna ambigüedad
- Si se han completado
- Si se pueden demostrar

2.6. MODELADO DEL ANÁLISIS, CASOS DE USO.

Diagrama de Casos de Uso

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Es una herramienta valiosa dado que es una técnica de aciertos y errores para obtener los requerimientos del sistema, justamente desde el punto de vista del usuario. Los diagramas de caso de uso modelan la funcionalidad del sistema usando actores y casos de uso. Los casos de uso son servicios o funciones provistas por el sistema para sus usuarios.

Símbolos de los casos de uso.

Sistema: El rectángulo representa los límites del sistema que contiene los casos de uso. Los actores se ubican fuera de los límites del Sistema.

Caso de uso: Se representan con óvalos. La etiqueta en el óvalo indica la función del sistema.

Actor: Un diagrama de caso de uso contiene los símbolos del actor y del caso de uso, junto con líneas conectoras. Los actores son similares a las entidades externas; existen fuera del sistema. El término actor se refiere a un rol específico de un usuario del sistema.

2.7. CONCEPTOS BÁSICOS DE LA ORIENTACIÓN A OBJETOS:

METODOLOGÍAS.

La orientación a objetos se basa en los siguientes conceptos elementales, que facilitan el abstraer los diferentes:

- Clase
- Objeto
- Atributo
- Método
- Herencia
- Polimorfismo
- Encapsulamiento

2.9. EL LENGUAJE DE MODELADO UNIFICADO (UML).

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, (Unified Modeling Lenguaje) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group), esta asociación se encarga de la definición y mantenimiento de estándares para aplicaciones de la industria de la computación. UML es un lenguaje gráfico que permite especificar, modelar, construir y documentar los elementos que forman un sistema software, principalmente orientado a objetos, sin embargo, UML no está diseñado exclusivamente para software orientado a objetos.

2.10. MODELADO ESTRUCTURAL.

CLASES

¿Qué es un diagrama de clases?

Un diagrama de clases muestra la existencia de clases y sus relaciones en el diseño lógico de un sistema. Un diagrama de clases puede representar todo o parte de la estructura de un sistema. Los diagramas de clase muestran la estructura de un modelo en particular, las entidades que deben existir, su estructura interna y las relaciones con otras clases. Los diagramas de clases no muestran información temporal.

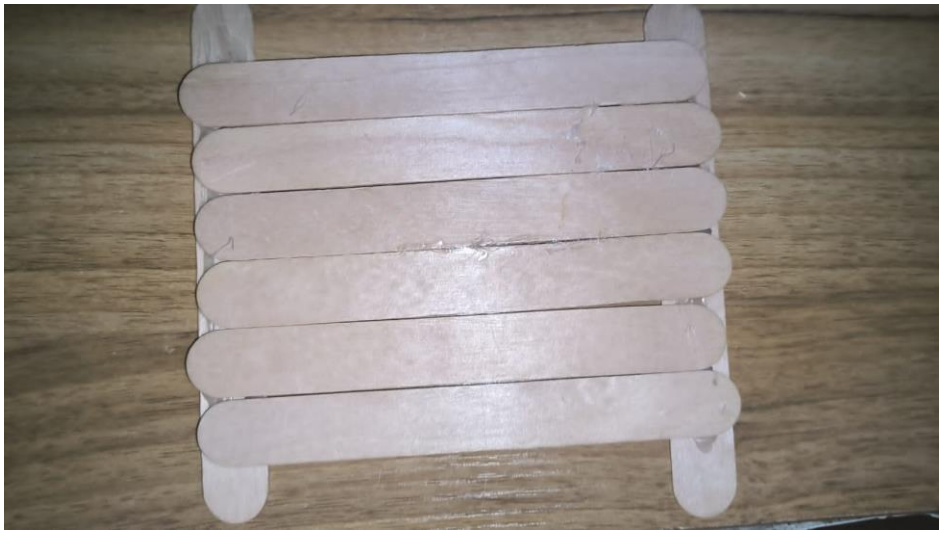
2.11. GRUPO DE DISEÑO DE CLASES EN PAQUETES.

Al identificar clases, éstas se deben agrupar en los paquetes para los propósitos de organización y configuración. El modelo de diseño se puede estructurar en unidades más pequeñas para hacerlo más comprensible. Agrupando los elementos modelo del diseño en los paquetes y los subsistemas, y mostrando cómo esas agrupaciones se relacionan una con otra, es más fácil entender la estructura total del modelo.

En conclusión, la Ingeniería del Software se erige como el bastión que impulsa le evolución tecnológica, ofreciendo un marco estructurado para la creación de soluciones informáticas efectivas. A lo largo de este análisis, hemos explorado los cimientos esenciales que sostienen esta disciplina, desde la planificación y el diseño hasta la implementación y el diseño hasta la implementación y mantenimiento. En un mundo cada vez más digital, la ingeniería del software no solo es un conjunto de herramientas y procesos, sino una expresión de la capacidad humana para innovar y mejorar la forma en que interactuamos con la tecnología. Al comprender y aplicar estos fundamentos, no solo creamos software funcional, sino que también contribuimos al progreso continuo de la sociedad digital.

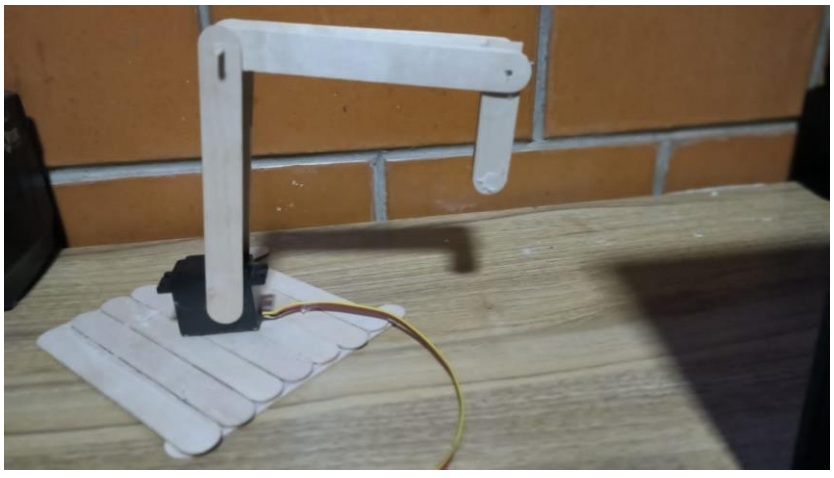
Fuentes de información:
https://plataformaeducativauds.com.mx/libro.php?idLibro=171018617216

Prototipo de Brazo Metálico.



BASE

ENSAMBLAMIENTO



PRUEBAS