Unidad I: Fundamentos de Ingenieria de Software

1.1. Conceptos básicos

a) Ingeniería

• Es la profesión en la que el conocimiento de las ciencias naturales y matemáticas obtenidas con el estudio, la práctica y la experiencia se aplica con juicio para desarrollar formas de utilizar de modo económico, los materiales y fuerzas de la naturaleza para beneficio de la humanidad

b) Software

• Es el conjunto de todos los programas que existen dentro de una computadora.

• Es el producto del desarrollo que realizan los ingenieros de software resultado de requerimientos de información.c) La Ingeniería de Software• Es una disciplina de la Ingeniería que comprende todos los aspectos de la producción del software desde las etapas iníciales de la especificación del sistema hasta el mantenimiento de éste después de que se libera.

La Ingeniería de Software incluye:

• Personas (quién lo hace)

·Proceso (la manera en que se hace)

·Proyecto (la realización)

·Producto (la aplicación de artefactos)

1.2. El papel evolutivo del software

El término fue introducido a fines del 60 y comienzo del 70, tras la crisis del software que se caracterizó por:

• Imprecisión en la planificación del proyecto y estimación de los costos.

• Baja calidad del Software.

• Dificultad de mantenimiento de programas con un diseño poco estructurado, etc.
En las décadas de 1980 y 1990 dos tendencias dominaron la ingeniería de software:
• El florecimiento explosivo de aplicaciones, incluyendo las de Internet.
• El Nacimiento de nuevas herramientas y paradigmas (formas de pensamiento, como la orientación a objetos).

Mitos del Software

Mitos: Son las creencias acerca del software y los procesos empleados para realizarlo.

- Mitos de la Administración

·Mitos del Cliente

·Mitos del Desarrollador

1.3. Etapas del desarrollo software

1) Investigación preliminar:

• Parte de una solicitud de requerimiento de un sistema de información, tiene tres partes:

a) Aclaración de la Solicitud

b) Estudio de Factibilidad: Técnica, Económica, Operacional

c) Aprobación de la Solicitud

2) Análisis de requerimientos:

Comprender todas las facetas importantes de la parte de la empresa bajo estudio:

a) ¿Qué es lo que hace?

b) ¿Cómo se hace?

c) ¿Con qué frecuencia se presenta?

d) Volumen de transacciones o decisiones

e) Grado de eficiencia de las tareas

f) ¿Existe algún problema?

g) ¿Qué tan serio y causa que lo origina?

3. Diseño del sistema:

• Plasma en un modelo los detalles que establecen la forma en la que el sistema cumplirá con los requerimientos identificados durante la fase de análisis

4. Desarrollo de Software:

• Se puede instalar software comprado (software genérico) o escribir programas diseñados a la medida del solicitante (software personalizado)

• La elección depende del costo, tiempo y disponibilidad de programadores.

5. Pruebas:

• En esta fase, el sistema se emplea de manera experimental para asegurarse que el software no tenga fallas, es decir, que funcione de acuerdo a las especificaciones del usuario y en la forma en que los usuarios esperan que lo haga.

6. Implementación:

Es el proceso de: Verificar e Instalar nuevo equipo, capacitar a usuarios, instalar la aplicación y dejar “montada” toda la infraestructura para su aplicación.

1.4. Clasificación de la tecnología en el desarrollo de software (Tecnología Estructurada y Orientada a Objetos)

Tecnología estructurada

Programación Estructurada es una técnica en la cual la estructura de un programa, esto es, la interpelación de sus partes realiza tan claramente como es posible mediante el uso de tres estructuras lógicas de control:

- ·Secuencia: Sucesión simple de dos o mas operaciones.
- ·Selección: bifurcación condicional de una o mas operaciones.
- ·Interacción: Repetición de una operación mientras se cumple una condición.

Estos tres tipos de estructuras lógicas de control pueden ser combinados para producir programas que manejen cualquier tarea de procesamiento de información.

Un programa estructurado está compuesto de segmentos, los cuales puedan estar constituidos por unas pocas instrucciones o por una página o más de codificación. Cada segmento tiene solamente una entrada y una salida, estos segmentos, asumiendo que no poseen lazos infinitos y no tienen instrucciones que jamas se ejecuten, se denominan programas propios. Cuando varios programas propios se combinan utilizando las tres estructuras básicas de control mencionadas anteriormente, el resultado es también un programa propio.

La programación Estructurada esta basada en el Teorema de la Estructura, el cual establece que cualquier programa propio (un programa con una entrada y una salida exclusivamente) es equivalente a un programa que contiene solamente las estructuras lógicas mencionadas anteriormente.

Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación. Esto es importante debido a que, es mucho más fácil comprender completamente el trabajo que realiza una función determinada, si todas las instrucciones que influyen en su acción están físicamente contiguas y encerradas por un bloque. La facilidad de lectura, de comienzo a fin, es una consecuencia de utilizar solamente tres estructuras de control y de eliminar la instrucción de desvío de flujo de control, excepto en circunstancias muy especiales tales como la simulación de una estructura lógica de control en un lenguaje de programación que no la posea.

VENTAJAS POTENCIALES

Un programa escrito de acuerdo a estos principios no solamente tendrá una estructura, sino también una excelente presentación.

Un programa escrito de esta forma tiende a ser mucho más fácil de comprender que programas escritos en otros estilos.

La facilidad de comprensión del contenido de un programa puede facilitar el chequeo de la codificación y reducir el tiempo de prueba y depuración de programas. Esto último es cierto parcialmente, debido a que la programación estructurada concentra los errores en uno de los factores más generador de fallas en programación: la lógica.

Un programa que es fácil para leer y el cual esta compuesto de segmentos bien definidos tiende a ser simple, rápido y menos expuesto a mantenimiento. Estos beneficios derivan en parte del hecho que, aunque el programa tenga una extensión significativa, en documentación tiende siempre a estar al día, esto no suele suceder con los métodos convencionales de programación.

La programación estructurada ofrece estos beneficios, pero no se la debe considerar como una panacea ya que el desarrollo de programas es, principalmente, una tarea de dedicación, esfuerzo y creatividad.

Tecnologia Orientada a Objetos

Lenguajes de Programación Orientado a Objetos

En 1985, E. Stroustrup extendió el lenguaje de programación C a C++, es decir C con conceptos de clases y objetos, también por esas fechas se creo desde sus bases el lenguaje EIFFEL.

En 1995 apareció el más reciente lenguaje OO, Java desarrollado por SUN, que hereda conceptos de C++.

El lenguaje de desarrollo más extendido para aplicaciones Web, el PHP 5, trae todas las características necesarias para desarrollar software orientado a objetos.

Además de otros lenguajes que fueron evolucionando, como el Pascal a Delphi.

Finalmente también otros lenguajes script como el ActionScript que si bien no es totalmente orientado a objetos pero sí posee las características.

La programación orientada a objetos es una de las formas más populares de programar y viene teniendo gran acogida en el desarrollo de proyectos de software desde los últimos años. Esta acogida se debe a sus grandes capacidades y ventajas frente a las antiguas formas de programar.

Hoy en día la tecnología orientada a objetos ya no se aplica solamente a los lenguajes de programación, además se viene aplicando en el análisis y diseño con mucho éxito, al igual que en las bases de datos. Es que para hacer una buena programación orientada a objetos hay que desarrollar todo el sistema aplicando esta tecnología, de ahí la importancia del análisis y el diseño orientado a objetos.

Ventajas:

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo
- Facilita el mantenimiento del software

Lo interesante de la POO es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

El modelo Orientado a Objetos

- Objetos
- Clases
- Herencia
- Envío de mensajes

1.5. Definición e historia de las herramientas CASE

Las [[herramientas CASE]] (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones

informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Es un sistema de software que intenta proporcionar ayuda automatizada a las actividades del proceso de software. Los sistemas CASE a menudo se utilizan como apoyo al método. La primera herramienta CASE como hoy la conocemos fue Exceleratoren 1984 , era para PC . Actualmente la oferta de herramientas CASE es muy amplia y tenemos por ejemplo el EASYCASE o WINPROJECT .

1.6. Clasificación de las herramientas CASE

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

1. Las plataformas que soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que cubren.
3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

La clasificación basada en las fases del ciclo de desarrollo cubre:

· Upper CASE (U-CASE), herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.

· Middle CASE (M-CASE), herramientas para automatizar tareas en el análisis y diseño de la aplicación.

· Lower CASE (L-CASE), herramientas que semi-automatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de Desarrollo rápido de aplicaciones.

Existen otros nombres que se le dan a este tipo de herramientas, y que no es una clasificación excluyente entre sí, ni con la anterior:

· Integrated CASE (I-CASE), herramientas que engloban todo el proceso de desarrollo software, desde análisis hasta implementación.

· MetaCASE, herramientas que permiten la definición de nuestra propia técnica de modelado, los elementos permitidos del meta modelo generado se guardan en un repositorio y pueden ser usados por otros analistas, es decir, es como si definiéramos nuestro propio UML, con nuestros elementos, restricciones y relaciones posibles.

· CAST (Computer-Aided Software Testing), herramientas de soporte a la prueba de software.

· IPSE (Integrated Programming Support Environment), herramientas que soportan todo el ciclo de vida, incluyen componentes para la gestión de proyectos y gestión de laconfiguración.

Por funcionalidad podríamos diferenciar algunas como:

- Herramientas de generación semiautomática de código.
- Editores UML.
- Herramientas de Refactorización de código.
- Herramientas de mantenimiento como los sistemas de control de versiones.