**ACTIVIDAD 4: ACTIVIDADES PROPUESTAS DEL LIBRO**

POR: SALVADOR MARTÍNEZ JIMÉNEZ

1.1 Buscar en la guía SWEBOK la clasificación de métodos de Ingeniería del Software.

En la guía SWEBOK (Software Engineering Body of Knowledge), los métodos de ingeniería de software se clasifican en las siguientes categorías:

**1. Métodos de Desarrollo Ágil:** Estos métodos se centran en la flexibilidad, la colaboración y la adaptación continua. Ejemplos incluyen Scrum, Kanban, Extreme Programming (XP) y Cristal.

**2. Métodos de Desarrollo Estructurado:** Estos métodos enfatizan la planificación y el diseño antes de la codificación. Ejemplos incluyen el Modelo en Cascada y el Modelo en V.

**3. Métodos de Desarrollo Iterativo e Incremental**: Estos métodos dividen el proceso de desarrollo en iteraciones, donde se agregan nuevas funcionalidades en cada ciclo. Ejemplos incluyen el Modelo Espiral y el RUP (Rational Unified Process).

**4. Métodos de Desarrollo Basados en Componentes:** Estos métodos se centran en la construcción de software a partir de componentes reutilizables. Ejemplos incluyen el Desarrollo Basado en Componentes (CBD) y la Arquitectura Orientada a Servicios (SOA).

**5. Métodos de Desarrollo Basados en Modelos:** En estos métodos, se utiliza la modelización para describir y diseñar el software antes de la implementación. Ejemplos incluyen el Desarrollo Dirigido por Modelos (MDD) y el Desarrollo Basado en Lenguajes Específicos de Dominio (DSL).

**6. Métodos de Desarrollo Basados en Prototipos:** Estos métodos involucran la creación de prototipos para entender y validar los requisitos antes de la implementación final. Ejemplos incluyen el Desarrollo Rápido de Aplicaciones (RAD) y el Desarrollo de Prototipos.

**7. Métodos de Desarrollo Basados en Pruebas**: En estos métodos, las pruebas son una parte fundamental del proceso de desarrollo. Ejemplos incluyen el Desarrollo Orientado a Pruebas (TDD) y el Desarrollo Basado en Pruebas (BDD).

**8. Métodos de Desarrollo Formal:** Estos métodos utilizan matemáticas y lógica para garantizar la corrección del software. Ejemplos incluyen la Especificación Formal y la Verificación Formal.

**9. Métodos de Desarrollo de Sistemas Empotrados y en Tiempo Real:** Estos métodos se centran en el desarrollo de software para sistemas embebidos y de tiempo real, que tienen requisitos estrictos de tiempo y rendimiento.

**10. Métodos de Desarrollo de Software de Código Abierto (Open Source):** Estos métodos involucran el desarrollo de software colaborativo y abierto, donde la comunidad de desarrolladores contribuye al proyecto.

**11. Métodos de Desarrollo de Software de Misión Crítica**: Estos métodos se aplican en el desarrollo de software crítico para la seguridad y la vida humana, como en la aviación y la atención médica.

Estas categorías representan diferentes enfoques y metodologías que los profesionales de la ingeniería de software pueden utilizar según las necesidades de un proyecto específico. La elección del método adecuado depende de factores como los requisitos del proyecto, el equipo de desarrollo y las restricciones de tiempo y recursos.

1.2 Analizar los diferentes tipos de actividades de Ingeniería del Software que un estudiante ha de llevar a cabo cuando desarrolla un programa para una práctica de laboratorio. Clasificarlos en función del artefacto que crea o modifica cada uno de los tipos de actividades identificados.

Cuando un estudiante desarrolla un programa para una práctica de laboratorio en el contexto de ingeniería de software, generalmente realiza una serie de actividades que involucran la creación o modificación de varios artefactos. A continuación, se presentan diferentes tipos de actividades clasificadas según los artefactos que se crean o modifican:

1. **Recolección de Requisitos:**

   - Artefactos: Documento de especificación de requisitos, lista de requisitos.

   - Actividades: Entrevistar al profesor o cliente (si es aplicable) para comprender los requisitos del laboratorio, analizar y documentar los requisitos en detalle.

2. **Diseño de Software:**

   - Artefactos: Diagramas de diseño, modelos UML, pseudocódigo.

   - Actividades: Diseñar la arquitectura del programa, crear diagramas de flujo, definir la estructura de datos y algoritmos necesarios.

3. **Implementación de Código:**

   - Artefactos: Código fuente del programa.

   - Actividades: Escribir el código del programa siguiendo el diseño propuesto, asegurarse de que cumpla con los requisitos.

4. **Pruebas de Software:**

   - Artefactos: Casos de prueba, resultados de prueba.

   - Actividades: Diseñar casos de prueba para verificar el funcionamiento correcto del programa, ejecutar pruebas, registrar resultados y corregir errores.

5. **Documentación del Software:**

   - Artefactos: Documentación técnica, manual de usuario.

   - Actividades: Crear documentación que explique cómo funciona el programa, cómo instalarlo y cómo utilizarlo, además de proporcionar comentarios en el código.

6. **Gestión de Configuración:**

   - Artefactos: Control de versiones del código fuente.

  -Actividades: Utilizar herramientas de control de versiones (como Git) para rastrear cambios en el código y asegurarse de que las versiones sean coherentes.

7. **Mantenimiento del Software:**

   - Artefactos: Código fuente actualizado, registros de cambios.

   - Actividades: Realizar actualizaciones, correcciones de errores y mejoras en el programa según sea necesario a lo largo del tiempo.

8. **Entrega y Despliegue:**

   - Artefactos: Archivos ejecutables, scripts de instalación.

   - Actividades: Preparar el programa para su entrega, crear un paquete de instalación (si es necesario) y realizar la instalación en los sistemas de laboratorio.

9**. Evaluación y Presentación:**

   - Artefactos: Informes de resultados, presentaciones.

   - Actividades: Evaluar el rendimiento del programa, preparar informes de resultados y presentar los resultados al profesor o evaluador.

Estas actividades son comunes en el proceso de desarrollo de software en un entorno de laboratorio académico. Cada una de ellas contribuye a la creación exitosa de un programa funcional y a menudo se realizan en secuencia o de manera iterativa, dependiendo de la metodología utilizada en el curso.

1.3 Leer el artículo seminal de Brooks sobre la complejidad del software (Brooks, 1987) es una actividad muy recomendable. Después de esa lectura, se proponen las siguientes actividades:

—Hacer un esquema de los elementos de la complejidad mencionados por Brooks.  
—Analizar esos elementos de complejidad mediante ejemplos cercanos.

**1. Complejidad de la Esencia:**

   - Ejemplo: Un sistema de reservas de vuelos en línea. La complejidad inherente radica en gestionar la disponibilidad en tiempo real de múltiples vuelos, aerolíneas y asientos, además de lidiar con cambios constantes debido a la compra de boletos, cancelaciones y cambios de horario.

**2. Variabilidad de Requisitos:**

   - Ejemplo:Un proyecto de desarrollo de una aplicación móvil. Los requisitos pueden cambiar a medida que los usuarios finales proporcionan retroalimentación, y los dispositivos móviles pueden tener especificaciones y capacidades variadas, lo que requiere adaptación constante.

**3. Dificultad en la Comunicación:**

   - Ejemplo: Un equipo de desarrollo de software que trabaja en un proyecto de aplicaciones empresariales. Los desarrolladores y los usuarios finales a menudo tienen diferentes comprensiones de los requisitos y la funcionalidad deseada, lo que puede llevar a malentendidos y errores de implementación.

**4. Mejoras Posibles (Accidentes):**

   - Ejemplo:La adopción de un sistema de control de versiones como Git. Esto mejora la colaboración y la gestión del código fuente, lo que reduce errores y permite a los equipos de desarrollo trabajar de manera más eficiente.

**5. Gestión de Proyectos:**

   - Ejemplo: La implementación de la metodología Scrum en un proyecto de desarrollo de software. Scrum ayuda a gestionar proyectos de manera más efectiva mediante iteraciones regulares, roles claros y comunicación constante, lo que mejora la gestión del tiempo y los recursos.

**6. Desarrollo de Herramientas:**

   - Ejemplo: La automatización de pruebas de software con herramientas como Selenium. Estas herramientas permiten a los equipos de desarrollo ejecutar pruebas de regresión de manera más eficiente, lo que reduce la carga de trabajo manual y mejora la calidad del software.

**7. Mejoras en las Prácticas:**

   - Ejemplo: La implementación de la integración continua (CI) en un equipo de desarrollo. La CI implica realizar pruebas automatizadas y compilaciones frecuentes, lo que ayuda a identificar y solucionar problemas de manera más rápida y efectiva.

Estos ejemplos ilustran cómo los elementos de complejidad mencionados por Brooks se aplican en situaciones reales de desarrollo de software y cómo las mejoras en la gestión, las herramientas y las prácticas pueden abordar estos desafíos.

1.5 El siguiente ejemplo está adaptado de una experiencia real reseñada en Navegapolis.net , un interesante blog sobre Ingeniería del Software.

«Miguel forma parte del equipo de programación de un sistema en entorno Visual Basic .NET. Ha trabajado durante dos días en una función —de ciento treinta líneas de código— capaz de comprobar la validez de una cadena como fecha, pero su función contiene dos errores: Considera como fechas válidas las cadenas vacías, y su cálculo de los años bisiestos no es correcto. [...] Miguel trabaja para una empresa que desarrolla software desde la perspectiva de los procesos, donde la premisa de producción industrial es que el valor del producto es resultado, en su mayor parte, de los procesos que siguen las personas que lo desarrollan. Así pues, asumiendo que su empresa sigue PSP (Personal Software Process), modelo que controla el tiempo, el tamaño del trabajo y el número de errores para medir la eficiencia y calidad de las personas y de los equipos, veámos cómo se determinaría la eficiencia del trabajo realizado por este trabajador. La información objetiva es clara: 130 líneas de código en 16 horas (2 días completos). Lo mismo ocurre con la calidad de su trabajo: dos errores en 130 líneas de código. Con estas informaciones los gestores de la empresa determinarán si Miguel es más o menos eficiente que la media, si la calidad de su trabajo está dentro del rango admisible, etc. Lo que no pueden descubrir es que Visual Basic incorpora de forma nativa la función IsDate() , y que si Miguel lo hubiera sabido, en lugar de dos días hubiera tardado un minuto en realizar el mismo trabajo, que además no contendría ningún error. Este hecho, basado en un caso real, pone de manifiesto cómo para algunas empresas es beneficioso este método de trabajo. La empresa que desarrolló este proyecto, concretamente, facturaba 250 dólares por hora de trabajo, por lo que programar esta función mediocre le aportó una facturación de 4.000 dólares»

Opinión:

Yo pienso que al programador le faltó una mejor capacitación para hacer un trabajo mas eficiente, lo mejor es como lo repasamos anteriormente, crear un plan basado en metodologías para hacer desde luego un trabajo de calidad y en un buen tiempo, garantizando así el puesto del programador y salvaguardar a la empresa.