Universidad Mariano Gálvez de Guatemala
Boca del Monte

Ingeniería en Sistemas. Ciclo II, “c”
Jornada Sábado.
ALGORITMOS
MIGUEL PICHIYA

Nombre: Luis Fernando Lima Ixcuná
Carné: 7690-20-17409

- **El ciclo de vida del software**

El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final, incluyendo su estado funcional. El propósito es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura que los métodos utilizados son apropiados. Estos métodos se originan en el hecho de que es muy costoso corregir los errores que se detectan tarde dentro de la fase de implementación (programación propiamente dicha), o peor aún, durante la fase funcional. En el modelo de ciclo de vida se intenta que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados. El ciclo de vida básico de un software consta de, al menos, los siguientes procedimientos:

- Análisis de requisitos, viabilidad de diseño y especificación de funciones definidas en lenguaje de programación.
- Análisis de la arquitectura en la creación, desarrollo, corrección e implementación del sistema.
- Pruebas en la integración de módulos, y subprograma(s) con cada conjunto o subconjunto.
- Pruebas beta o de validación que garanticen que en el procedimiento de ejecución del software se cumple con todas las especificaciones originales.
- Mantenimiento de corrección de errores y restricciones.
- Documentación de toda la información.

El orden y la presencia de cada uno de estos procedimientos dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores. En el caso del software libre se tiene un ciclo de vida mucho más dinámico, puesto que muchos programadores trabajan en simultáneo desarrollando sus eliminaciones.

- **¿Qué es un Análisis de Requerimientos?**

Un análisis de requerimientos es un estudio profundo de una necesidad tecnológica que tiene una empresa, organización o negocio. En este proceso, se realiza un análisis exhaustivo del sistema que se va a desarrollar. Se definen y aplican técnicas que permitan analizar los requisitos necesarios para su buen desarrollo. De esta forma, se logra reconocer y entender cuáles son las verdaderas necesidades que el sistema debe solucionar.
Un análisis de requerimientos:

- Realiza un estudio profundo de la necesidad tecnológica que tiene el negocio.
- Especifica las características operacionales que tendrá el software a desarrollar.
- Tiene en cuenta las diferentes áreas de trabajo: reconocimiento del problema, evaluación, modelado, especificación y revisión.
- Realiza a través de entrevistas, talleres, observación, indagación, revisión documental y demás técnicas específicas.

- Describe el plan del proyecto a seguir.
- Es fundamental entregar el proyecto dentro del tiempo y presupuesto acordados y de los objetivos de negocio.

Características de un buen análisis de requerimientos

- Análisis completo: se deben reflejar todos los requerimientos, necesidades y especificaciones de la forma más exhaustiva y definida posible.
- Consistente: que no pueda generar dudas ni contradicciones y que tenga coherencia a lo largo del tiempo.
- Claro: esto hace referencia a la redacción, la cual debe ser clara para evitar posibles malinterpretaciones.
- Posibilidad de verificación: que se puedan comprobar los datos reflejados y así revisar si se están cumpliendo con los requisitos definidos. Es un paso muy importante para validar el análisis.
- Priorizable: debe permitir una organización jerárquica por prioridades, en función de su relevancia. Se pueden clasificar por esenciales, urgentes, opcionales, etc.
- Fácilmente modificable: que permite la modificación a lo largo del tiempo para ir optimizando los requerimientos.
- Proporciona un mapa para llegar con éxito al desarrollo del software o app.

- **Diseño de software**

Diseño de software es el proceso de diseño para la planificación de una solución de software. Este proceso es, por regla general, necesario para que los programadores puedan manejar la complejidad que la mayoría de los programas informáticos poseen y para disminuir el riesgo de desarrollos erróneos.

- Ingeniería de requerimientos

Generalmente, cliente y contratista analizan primero los requerimientos que resultan, desde el punto de vista del cliente, para el software a diseñar. En este contexto, el cliente prepara el así llamado pliego de condiciones.

- Realización

A continuación, cliente y contratista elaboran un concepto, en el que se define con qué estructuras de programa, técnicas de programación y algoritmos los requerimientos analizados anteriormente se deben cumplir y programar. El contratista especifica los resultados de este concepto en el denominado pliego de condiciones.

- **Implementación**

En TI, implementación significa la distribución o provisión de software. La implementación se realiza a través de procesos automatizados, que se utilizan para instalar y configurar las soluciones de software. La implementación o implementación de software se refiere a los procesos en su mayoría semiautomáticos o completamente automáticos de distribución de software, especialmente en las empresas. El despliegue incluye aspectos como la

instalación, configuración, actualización y mantenimiento de sistemas operativos y sistemas de aplicación en PC o servidores. Las actualizaciones y parches, así como su provisión, también forman parte de la implementación.

Las empresas y organizaciones más grandes, en particular, utilizan una distribución de software profesional y en su mayoría centralizada. Esto se hace a menudo mediante el uso de scripts de implementación y soluciones de software independientes. Por lo general, la instalación no requiere ninguna interacción adicional por parte del usuario. Para organizaciones más pequeñas y áreas de aplicación, un solo administrador o un empleado capacitado puede llevar a cabo la instalación y distribución. La tarea de la implementación y la estrategia de implementación es un diseño eficiente de la gestión operativa de TI.

- **Pruebas**

Las pruebas de integración dentro del software testing chequean la integración o interfaces entre componentes, interacciones con diferentes partes del sistema, como un sistema operativo, sistema de archivos y hardware o interfaces entre sistemas. Las pruebas de integración son un aspecto clave del software testing. Es esencial que un probador de software tenga una buena comprensión de los enfoques de prueba de integración, para lograr altos estándares de calidad y buenos resultados.

- Tipos De Pruebas De Integración Dentro del Sofware Testing

Dentro del software testing existen muchos tipos o enfoques diferentes para las pruebas de integración. Los enfoques más populares y de uso frecuente son las pruebas de integración Big Bang, las pruebas de integración descendente, las pruebas de integración ascendente y las pruebas de integración incremental.La elección del enfoque depende de varios factores como el costo, la complejidad, la criticidad de la aplicación, etc.

Hay muchos tipos menos conocidos de pruebas de integración, como la integración de servicios distribuidos, las pruebas de integración sándwich, la integración de la red troncal, la integración de alta frecuencia, la integración de capas, etc.

- Prueba De Integración Big Bang

En las pruebas de integración de Big Bang, todos los componentes o módulos se integran simultáneamente, después de lo cual todo se prueba como un todo.

Ventaja: Las pruebas de Big Bang tienen la ventaja de que todo está terminado antes de que comiencen las pruebas de integración.

Desventaja: La principal desventaja es que, en general, lleva mucho tiempo y es difícil rastrear la causa de las fallas debido a esta integración tardía.

- Prueba De Integración Descendente

Las pruebas se llevan a cabo de arriba a abajo, siguiendo el flujo de control o la estructura arquitectónica (por ejemplo, comenzando desde la GUI o el menú principal). Los componentes o sistemas se sustituyen por stubs.

Ventajas: El producto probado es muy consistente porque la prueba de integración se realiza básicamente en un entorno casi similar al de la realidad. Los códigos auxiliares se pueden escribir en menos tiempo porque en comparación con los controladores, los códigos auxiliares son más sencillos de crear.

Desventajas: La funcionalidad básica se prueba al final del ciclo. Prueba De Integración Ascendente

Las pruebas se llevan a cabo desde la parte inferior del flujo de control hacia arriba. Los componentes o sistemas se sustituyen por controladores.

- Ventaja del enfoque ascendente:

En este enfoque, el desarrollo y las pruebas se pueden realizar juntos para que el producto o la aplicación sea eficiente y de acuerdo con las especificaciones del cliente.

Desventajas del enfoque ascendente:

Podemos detectar los defectos de la interfaz clave al final del ciclo

- **Mantenimiento de software**

En ingeniería del software, el mantenimiento de software es la modificación de un producto de software después de la entrega, para corregir errores, mejorar el rendimiento, u otros atributos.1 El mantenimiento del software es una de las actividades más comunes en la ingeniería de software. El mantenimiento de software es también una de las fases en el ciclo de vida de desarrollo de sistemas (SDLC, sigla en inglés de system development life cycle), que se aplica al desarrollo de software. La fase de mantenimiento es la fase que viene después del despliegue (implementación) del software en el campo.

Una percepción común del mantenimiento es que se trata meramente de la corrección de defectos. Sin embargo, un estudio indicó que la mayoría, más del 80%, del esfuerzo de mantenimiento es usado para acciones no correctivas (Pigosky 1997). Esta percepción es perpetuada por usuarios enviando informes de problemas que en realidad son mejoras de funcionalidad al sistema. El mantenimiento del software y la evolución de los sistemas fue abordada por primera vez por Meir M. Lehman en 1969. Durante un período de veinte años, su investigación condujo a la formulación de las leyes de Lehman (Lehman 1997). Principales conclusiones de su investigación incluyen que el mantenimiento es realmente un desarrollo evolutivo y que las decisiones de mantenimiento son ayudadas por entender lo que sucede a los sistemas (y al software) con el tiempo. Lehman demostró que los sistemas continúan evolucionando con el tiempo. A medida que evolucionan, ellos crecen más complejos a menos que se toman algunas medidas como refactorización de código para reducir la complejidad.

Los problemas claves de mantenimiento de software son administrativos y técnicos. Problemas clave de administración son: alineación con las prioridades del cliente, dotación de personal, cuál organización hace mantenimiento, estimación de costos. Son cuestiones técnicas claves: limitado entendimiento, análisis de impacto, pruebas (testing), medición de mantenibilidad.

El mantenimiento de software es una actividad muy amplia que incluye la corrección de errores, mejoras de las capacidades, eliminación de funciones obsoletas y optimización. Debido a que el cambio es inevitable, se debe desarrollar mecanismos para la evaluación, controlar y hacer modificaciones. Así que cualquier trabajo realizado para cambiar el software después de que esté en operación es considerado trabajo de mantenimiento. El propósito es preservar el valor del software sobre el tiempo. El valor puede ser mejorado ampliando la base de clientes, cumpliendo requisitos adicionales, siendo cada vez más fácil de usar, más eficiente y empleando más nuevas tecnología. El mantenimiento puede abarcar 20 años, mientras que el desarrollo puede estar entre 1 y 2 años.

- **Estilos**

La programación es el proceso de crear un conjunto de instrucciones que le dicen a una computadora como realizar algún tipo de tarea. Pero no solo la acción de escribir un código para que la computadora o el software lo ejecute. Incluye, además, todas las tareas necesarias para que el código funcione correctamente y cumpla el objetivo para el cual se escribió. 1

En la actualidad, la noción de programación se encuentra muy asociada a la creación de aplicaciones de informática y videojuegos. En este sentido, es el proceso por el cual una persona desarrolla un programa, valiéndose de una herramienta que le permita escribir el código (el cual puede estar en uno o varios lenguajes, como C++, Java y Python, entre muchos otros) y de otra que sea capaz de “traducirlo” a lo que se conoce como lenguaje de máquina, que puede "comprender" el microprocesador.2

- **Diseño de algoritmos**

En ciencias de la computación, el diseño de algoritmos es un método específico para poder crear un modelo matemático ajustado a un problema específico para resolverlo. El diseño de algoritmos o algorítmica es un área central de las ciencias de la computación, también muy importante para la investigación de operaciones (también conocida como investigación operativa), en ingeniería del software y en otras disciplinas afines.

- **Testing**

¿Qué es el software testing?

El testing de software o software QA, es un proceso para verificar y validar la funcionalidad de un programa o una aplicación de software con el objetivo de garantizar que el producto de software esté libre de defectos. La intención final es que coincida con los requisitos esperados para entregar un producto de calidad. Implica la ejecución de componentes de software o sistema utilizando herramientas manuales o automatizadas para evaluar una o más propiedades de interés. El testing de software es un proceso paralelo al desarrollo de software cuyas tareas deben ir realizándose a medida que se construye el producto para evitar problemas en la funcionalidad de manera previa a su lanzamiento.

- ¿Por qué es importante el software testing?

Las pruebas de software son importantes porque permiten identificar de manera temprana si hay algún problema en el software, facilitando su resolución antes de la entrega del producto. Un producto de software debidamente probado garantiza calidad, seguridad, confiabilidad y alto rendimiento, además de otros beneficios como ahorro de tiempo, seguridad y satisfacción del cliente.
Objetivos del testing de software

- Detectar y corregir errores.
- Proporcionar calidad y confiabilidad del software.
- Asegurar la correcta funcionalidad del producto.
- Evitar futuros errores.
- Facilitar la toma de decisiones para pasar a producción los desarrollos que no contengan errores.
- Cumplir con los requisitos del negocio y satisfacción del usuario.
- Evitar la aparición de nuevos defectos en el futuro que puedan afectar al software.