Introducción a Calidad de Software

Problema, especificación, algoritmo, programa.

Dado un problema a resolver (de la vida real), queremos:

- Poder **describir** de una manera clara y unívoca (*especificación*): esta descripción debería poder ser **validada** contra el problema real.
- Poder **diseñar** una solución acorde a dicha especificación: este diseño debería poder ser **verificado** con respecto a la especificación
- Poder **implementar** un programa acorde a dicho diseño: este programa debería poder ser **verificado** con respecto a su especificación y su diseño (debería ser la *solución* al problema planteado)

Validación y Verificación

En el contexto de la ingeniería de software, verificación y validación (**V&V**) es el proceso de comprobar que un sistema de software cumple con sus especificaciones y que cumple su propósito previsto. También puede ser
denominado como el control de la **calidad del software**.

VALIDACIÓN: ¿Estamos haciendo el producto correcto? El software debería hacer lo que el usuario requiere de él.

VERIFICACIÓN:¿Estamos haciendo el producto correctamente? El software debería realizar lo que su especificación indica.

La calidad del software se mide, generalmente, en atributos de calidad:

- Confiabilidad: Se refiere a la capacidad del software para realizar sus funciones de manera confiable y predecible. Evalúa la estabilidad, la disponibilidad, la tolerancia a fallos y la capacidad de recuperación del software en caso de errores o problemas.
- Corrección: la corrección en la calidad del software se refiere a la capacidad del software para funcionar sin errores y de acuerdo con los requisitos establecidos, lo cual es fundamental para garantizar su utilidad y confiabilidad.
- Facilidad de mantenimiento: Se refiere a la facilidad con la que el software se puede mantener y modificar. Evalúa la estructura del código, la modularidad, la claridad, la documentación y la facilidad para agregar nuevas funcionalidades o corregir errores.
- Reusabilidad:se refiere a la capacidad de un componente, módulo o sistema de software para ser utilizado en múltiples contextos o aplicaciones diferentes, sin necesidad de realizar modificaciones extensas o reescribirlo desde cero. Implica diseñar y desarrollar componentes de software que

sean fácilmente extraíbles y reutilizables en diferentes proyectos, promoviendo la eficiencia, la productividad y la calidad del software.

- Eficiencia: se refiere al rendimiento y la eficiencia en el uso de recursos del software. Evalúa la velocidad de ejecución, el consumo de memoria, el uso del procesador y otros aspectos relacionados con el rendimiento del software.
- Verificabilidad + claridad: El atributo de verificabilidad en calidad de software se refiere a la capacidad de verificar y confirmar de manera objetiva que el software cumple con los requisitos establecidos y se comporta como se espera. Es la medida de cuán fácil es evaluar y determinar si el software es correcto, confiable y funciona según lo previsto. El atributo de claridad en calidad de software se refiere a la legibilidad, comprensibilidad y transparencia del código fuente y la documentación del software. Implica que el software se presenta de manera clara, bien estructurada y fácil de entender para los desarrolladores, usuarios y otros interesados.
- Usabilidad: Se refiere a la facilidad de uso del software y a la experiencia del usuario. Evalúa la interfaz de usuario, la navegación, la claridad de las instrucciones y la capacidad del software para ser aprendido y utilizado de manera eficiente.
- Robustez: El atributo de robustez en calidad de software se refiere a la capacidad de un software para mantener su funcionalidad y desempeño incluso en situaciones adversas o inesperadas, como entradas inválidas, errores de usuario, condiciones de carga intensiva o fallos en el entorno de ejecución. Un software robusto debe ser capaz de recuperarse de errores, manejar excepciones, mantener la integridad de los datos y evitar el colapso del sistema.
- Seguridad :Se refiere a la protección de los datos y la prevención de accesos no autorizados o vulnerabilidades. Evalúa las medidas de seguridad implementadas en el software, como el cifrado, la autenticación, el control de acceso y la gestión de riesgos.
- Funcionalidad : Se refiere a la capacidad del software para cumplir con los requisitos funcionales establecidos. Evalúa si el software realiza las funciones previstas y si se comporta correctamente en diferentes situaciones y escenarios.
- Interoperabilidad: se refiere a la capacidad de un sistema o software para interactuar y comunicarse de manera efectiva con otros sistemas, componentes o aplicaciones. Implica la capacidad de intercambiar información, compartir recursos y cooperar con otros sistemas de manera transparente y sin problemas.
- ETC.

Estos son solo algunos de los atributos de calidad comunes que se miden en la evaluación del software. La importancia y el enfoque de estos atributos pueden variar según el contexto y los requisitos específicos del software y los usuarios finales. Es fundamental considerar y equilibrar estos atributos durante el desarrollo y la evaluación del software para garantizar su calidad general.

Nociones básicas y proceso de V&V

- Falla: diferencia entre los resultados esperados y reales. Es la manifestación del defecto.
- Defecto: desperfecto en algún componente del sistema (en el texto del programa, una especificación, un diseño, etc). Origina cero, una o más fallas.

- Error: equivocación humana. Lleva a uno o más defectos presentes en un producto de software.

V&V debería aplicarse a cada instancia del proceso de desarrollo. No sólo el código, sino también todos los subproductos generados durante el desarrollo deben ser sometidos a actividades de V&V.

Los **objetivos** principales son:

- Descubrir defectos en el sistema.
- Asegurar que el software respeta su especificación.
- Determinar si satisface las necesidades de sus usuarios.

Las **metas** de la V&V son:

- La verificación y la validación deberían establecer la confianza: esto no significa que esté completamente libre de defectos, sino que debe ser lo suficientemente bueno para su uso previsto y el tipo de uso determinará el grado de confianza que se necesita.

La validación se centra en "hacer el software correcto", asegurándose de que el software sea adecuado para su uso previsto y satisfaga las necesidades del usuario.

La verificación se enfoca en garantizar que se haya construido correctamente el software, mientras que la validación se centra en garantizar que el software sea el adecuado y cumpla con las expectativas del usuario final. Ambos procesos son esenciales para asegurar la calidad y la confiabilidad del software.

Verificación estática y dinámica

La **verificación estática** se basa en el análisis del código y la estructura del software **sin ejecutarlo**. No se requiere la ejecución del programa para identificar problemas potenciales. Se centra en la revisión del código fuente, el diseño y la documentación para detectar errores, violaciones de estándares y posibles problemas.

Ejemplos de técnicas de verificación estática incluyen: revisión de código, análisis estático, revisión de diseño, análisis de flujo de datos, análisis de dependencia, entre otros.

La **verificación dinámica** se basa en la ejecución del software y la observación de su comportamiento durante la ejecución. Implica el uso de pruebas y análisis en tiempo de ejecución para verificar si el software cumple con los requisitos y las expectativas establecidas.

Ejemplos de técnicas de verificación dinámica incluyen: pruebas funcionales, pruebas de rendimiento, pruebas de estrés, pruebas de usabilidad, pruebas de seguridad, entre otros.

En resumen, la verificación estática se centra en el análisis del código y la estructura sin ejecutar el software, mientras que la verificación dinámica se basa en la ejecución del software y la observación de su comportamiento durante la ejecución. Ambas técnicas son importantes y complementarias en el proceso de verificación de software.

Testing

¿Qué es hacer testing?
Es el proceso de ejecutar un producto para:
-Verificar que satisface los requerimientos (especificación)
-Identificar diferencias entre el comportamiento real y el comportamiento esperado.

Su objetivo es encontrar defectos en el software. Representa entre el 30% y 50% del costo de un software confiable.

Niveles de Test

- Test de sistema: Comprende todo el sistema. Por lo general constituye el test de aceptación
- Test de integración: Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto. Testeamos la interacción, la comunicación entre partes
- Test de unidad: Se realiza sobre una unidad de código pequeña, claramente definida.

Términos de testing

- Programa bajo test : Es el programa que queremos saber si funciona bien o no.
- Test Input (o dato de prueba): Es el conjunto de datos o valores de entrada que se utilizan como entrada para ejecutar un caso de prueba. Las entradas de prueba se seleccionan de manera que cubran diferentes escenarios y condiciones para evaluar el comportamiento del programa bajo test en diversas situaciones.
- Test Case: Caso de Test (o caso de prueba). Es un programa que ejecuta el programa bajo test usando un dato de test, y chequea (automáticamente) si se cumple la condición de aceptación sobre la salida del programa bajo test. Un caso de prueba tiene un objetivo claro y se enfoca en evaluar una característica o aspecto particular del software. Puede incluir datos de entrada, acciones a realizar, condiciones previas y resultados esperados.
- Test Suite: Es un conjunto de casos de Test (o de conjunto de casos de prueba). Una suite de pruebas puede contener múltiples casos de prueba que cubren diferentes aspectos del software. La idea de una test suite es proporcionar una cobertura exhaustiva de las funcionalidades y asegurarse de que se prueben todas las condiciones y escenarios relevantes.

Durante el proceso de testing, se crean casos de prueba que consisten en definir los datos de entrada y las acciones a realizar para evaluar una funcionalidad específica del programa bajo test. Estos casos de prueba se agrupan en una test suite, que es una colección de pruebas relacionadas que se ejecutan en conjunto para verificar el correcto funcionamiento del software.

Preguntas del proceso de testing

- ¿Cuál es el programa de test?
  Es la implementación de una especificación.

- ¿Entre qué datos de prueba puedo elegir?
  Aquellos que cumplen la precondición (requieres) en la especificación.
  -¿Qué condición de aceptación tengo que chequear?
  La condición que me indica la postcondición (aseguras) en la especificación.
  -¿Qué pasa si el dato de prueba no satisface la precondición de la especificación?
  Entonces no tenemos ninguna condición de aceptación (¡No es necesario chequear nada!)

Limitaciones del testing

Al no ser exhaustivo, el testing NO puede probar (demostrar) que el software funciona correctamente.

El testing puede demostrar la presencia de errores, nunca su ausencia (Dijkstra)

Una de las mayores dificultades es encontrar un conjunto de tests adecuado, que sea **suficientemente grande** para abarcar el dominio (datos que cumplen el requiere) y maximizar la probabilidad de encontrar errores, y que sea **suficientemente pequeño** para poder ejecutar el proceso con cada elemento del conjunto y minimizar el costo del testing.

Seleccionar datos de test

Por el tratamiento que reciben, hay inputs que son *parecidos entre sí* por lo que probar el programa con uno equivaldría a probarlo con cualquier otro de estos. Para definir que dos inputs son *parecidos entre sí* si únicamente disponemos de la especificación, hacemos uso de nuestra *experiencia* y utilizamos la *intuición*.

No hay un algoritmo que proponga casos tales que encuentren todos los errores en cualquier programa. Ninguna técnica puede ser efectiva para detectar todos los errores en un programa arbitrario

- **Test de Caja negra**
  Los casos de test se generan analizando la especificación sin considerar la implementación.
  Los datos de test se derivan a partir de la descripción del programa sin conocer su implementación

El testing de caja negra se centra en probar el software desde una perspectiva externa, sin conocer los detalles internos de su funcionamiento. En este enfoque, el tester no tiene acceso al código fuente ni a la estructura interna del sistema. En su lugar, se concentra en analizar la funcionalidad del software y su comportamiento en relación con las entradas y salidas esperadas. El objetivo es descubrir errores o defectos en la funcionalidad y asegurarse de que el software cumple con los requisitos y expectativas del usuario. Se utilizan técnicas como pruebas de casos de uso, pruebas de equivalencia, pruebas de límites y pruebas aleatorias para realizar pruebas exhaustivas sin tener en cuenta la implementación interna.

- **Test de Caja blanca**
  Los casos de test se generan analizando la
  implementación para determinar los casos de test.
  Los datos de test se derivan a partir de la estructura interna del programa.

El testing de caja blanca se centra en probar el software desde una perspectiva interna, teniendo acceso al código fuente y a la estructura interna del sistema. En este enfoque, el tester examina la lógica interna del software, las estructuras de datos y los flujos de control para identificar y probar diferentes caminos de ejecución. Se utilizan técnicas como pruebas de cobertura de código, pruebas de flujo de control y pruebas de bucles para garantizar que todas las partes del código sean probadas y verificar que el software funciona correctamente desde un punto de vista interno.