Notas

Video 10 Code Style

el estilo de codificación juega un papel importante en la legibilidad y comprensión del código. Al trabajar en equipos de desarrollo, es esencial tener pautas y reglas de estilo de codificación para mantener la consistencia en todo el proyecto.

El estilo puede ser por ejemplo: la nomenclatura, es importante utilizar nombres de variables y rutinas que sean distintos entre sí y reflejen claramente su propósito. Se debe nombrar las clases y las instancias de objetos de manera distinta para facilitar la comprensión del código. En la nomenclatura de archivos, se recomienda evitar el uso de espacios en los nombres de archivos, ya que esto puede causar problemas al trabajar en diferentes plataformas.

Sobre la identacion donde se discute sobre la sangría, las opciones de usar espacios o tabs para indentar el código. Pero es más importante es mantener la consistencia en la sangría y utilizar una convención que facilite la comprensión del código.

Cada estilo tiene sus propias reglas sobre la colocación de las llaves, si se elige uno se debe mantener la consistencia en todo el código.

el espacio en blanco puede afectar la comprensión del código. El espaciado puede hacer que la interpretación del código sea ambigua.

video 11 Code Style Examples

El video muestra una serie de ejemplos donde muestra ejemplos de estilo de codificación y cómo puede afectar la apariencia, legibilidad y comprensión del código. Muestra porque es importante estilo de codificación para las personas que leen el código, ya que la máquina no percibe los espacios en blanco de la misma manera. Se ven ejemplos como el espaciado entre operadores, la importancia de la sangría para mostrar la estructura del código y la auto-documentación. Muestra que la longitud de los nombres de variables no tiene mucho impacto en el rendimiento, pero es crucial para la legibilidad. Lo más importante es consistencia en el estilo de codificación

video 12 Debuggin

los errores en el código son comunes y que la depuración es la forma de eliminarlos. La depuración de código y presenta dos enfoques principales:

- la impresión de mensajes de error y el uso de herramientas de depuración. imprimir mensajes de error en diferentes partes del código puede ayudar a identificar la ubicación del problema, pero los mensajes de error son difíciles de interpretar. la impresión de variables y el uso de métodos de depuración pueden ser propensos a errores
- las herramientas de depuración permiten establecer puntos de interrupción en el código y detener la ejecución en esos puntos, esto permite examinar el estado del programa para investigar el problema. las herramientas de depuración son útiles y ofrecen muchas funciones útiles, pero también se tiene que estar familiarizado y se debe tener cuidado para evitar errores adicionales.

video 13 Static Analysis

El análisis estático es una técnica de programación que utiliza herramientas para identificar y describir errores comunes en el código. A diferencia de los compiladores, el análisis estático examina el código en reposo, sin necesidad de ejecutarlo, para encontrar problemas propensos a errores.

el análisis estático también puede evaluar el estilo de código y otras características relacionadas con la calidad. Aunque pueden existir falsos positivos, estas herramientas proporcionan una forma rápida de identificar posibles áreas problemáticas y mejorar la calidad del software.

video 14 Comenting

debido a que muchos desarrolladores tienden a olvidar el código que han escrito en un corto período de tiempo, esto dificulta su mantenimiento y actualización. Por lo que es importante de tener una documentación en el desarrollo de software; se debe escribir comentarios que expliquen el propósito y funcionamiento del código, ya que esto facilita su comprensión y evita problemas futuros.

video 15 Version Control Systems

los sistemas de control de versiones, como Git, gestionan el código fuente de los proyectos de software. Estos sistemas permiten realizar copias de seguridad, controlar los cambios, trabajar en ramas de desarrollo y características, y fusionar los cambios realizados por varios desarrolladores en una base de código única, además, facilitan la trazabilidad y la resolución de conflictos de fusión.

Los repositorios remotos, como GitHub y GitLab, permiten compartir y colaborar en el código, utilizando funciones como solicitudes de extracción y revisiones de código.

video 16 Build Process

las herramientas de construcción son utilizadas en el desarrollo de software para automatizar el proceso de compilación y construcción de proyectos. estas herramientas permiten definir objetivos y dependencias, lo que significa que se pueden especificar qué archivos o tareas deben ser construidos y en qué orden, herramientas como:

- **Make** es la herramienta principal, pero su sintaxis puede ser críptica para los no familiarizados
- **Apache Ant** mejora esta sintaxis y proporciona una herramienta más aceptada y robusta para controlar el proceso
- **Maven** utiliza un archivo de configuración más simple y legible
- **Gradle** busca combinar los beneficios de Maven y Ant con un enfoque basado en dependencias.

estas herramientas permiten configurar comandos, macros y dependencias para tener un mayor control sobre el proceso de construcción.

video 17 Test Selection

La selección de pruebas es el proceso de elegir los casos de prueba que se ejecutarán contra el software para descubrir esos defectos. El objetivo de las pruebas es encontrar defectos en el software, desde los requisitos hasta el código implementado.

Es importantes tener un oráculo, que determina si una prueba pasa o no al comparar la salida real con la salida esperada. Los humanos suelen ser el oráculo, pero también se pueden utilizar oráculos automatizados.

La cobertura del código es una medida de qué tan bien funcionan las pruebas y se refiere a la cantidad de código ejecutado durante las pruebas. (declaración, condición y decisión)

Es importancia de tener pruebas automatizadas en lugar de depender solo de pruebas manuales realizadas por los desarrolladores.

video 18 Test Adequacy

encontrar errores no es suficiente para considerar que las pruebas son buenas, ya que no existe una forma exhaustiva de probar un programa. La prueba exhaustiva es teóricamente imposible debido a la infinita cantidad de entradas, por lo que podemos medir qué tan bien se cubren aspectos como la estructura, los insumos y los requisitos del programa.

Existen varios criterios de educación, pero cumplir con estos criterios no garantiza que el código sea correcto, es posible engañar a la cobertura del código agregando pruebas simples, pero poco útiles.

las pruebas efectivas son más que criterios de cobertura y deben ser capaces de encontrar nuevos errores, proporcionar seguridad adicional, ser robustas ante cambios en el programa y ser razonables en cuanto a los recursos disponibles.

video 19 Test-Driven Development

El desarrollo basado en pruebas se presenta como un enfoque más productivo, donde se prioriza lo que el software debe hacer a cómo se implementa.

Se propone invertir el orden de implementación: en lugar de codificar primero y luego probar, se sugiere escribir las pruebas antes y compilar el código hasta que las pruebas pasen. Esto permite identificar el progreso del desarrollo a medida que se superan diferentes aspectos de las pruebas, incluso si otras pruebas aún no pasan.

Con esta metodología, se obtienen actualizaciones incrementales de calidad en el código, lo cual es beneficioso para los gerentes y se logra un código final de mejor calidad. cuando se alcanza el 100% de pruebas pasadas, se considera que el proceso de codificación ha finalizado.

video 20 Continuous Integration

la integración continua permite eliminar tareas simples para que las personas se centren en aspectos más importantes.

La integración continua automatizada permite el rechazo de confirmaciones que no cumplen los estándares y la revisión del código antes de su integración. Esto garantiza una evaluación adicional y una decisión conjunta sobre qué confirmaciones se incluirán en el código compartido.

la integración continua automatizada mejora la eficiencia y la calidad del desarrollo al eliminar tareas manuales y brindar un enfoque sistemático y controlado para garantizar la integridad del código.

Ideas Principales

Las ideas principales abarcan: el estilo de codificación para garantizar la legibilidad y consistencia del código, la depuración, el análisis estático se destaca como una herramienta útil para identificar errores comunes y mejorar la calidad del software, los comentarios y documentación. Los sistemas de control de versiones permiten gestionar el código fuente y colaborar en proyectos, las herramientas de construcción automatizan el proceso de compilación y construcción de proyectos, las pruebas, se mencionan aspectos como la selección de pruebas, la cobertura de código y la adecuación de las pruebas, integración continúa automatizada como un enfoque que mejora la eficiencia y calidad del desarrollo

Resumen

Todos los aspectos que se exploraron en esta lista de videos nos ayudan a garantizar la legibilidad, consistencia y calidad del código, así como la eficiencia y confiabilidad en el proceso de desarrollo. Al enfocarnos en estos aspectos, podemos lograr un software de mayor calidad, más fácil de mantener y mejorar continuamente.