Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología Internet 2 U-2018

Pre-Laboratorio 1

Evaluador:
Víctor Arica

Nombre:
Ricardo Baptista CI 24.463.392

Caracas, 23 de diciembre de 2018

Preguntas:

1) Defina la diferencia entre un Sistema de Control de Versiones Centralizado y Distribuido.

R:

El control de versiones distribuido toma un enfoque entre iguales (peer-to-peer), opuesto al enfoque de cliente-servidor de los sistemas centralizados. En lugar de un único repositorio central en el cual los clientes se sincronizan, la copia local del código base de cada *peer* es un repositorio completo. El control de versiones distribuido sincroniza los repositorios intercambiando ajustes (conjuntos de cambios) entre iguales. Esto establece algunas diferencias importantes en comparación a un sistema centralizado:

- No existe una copia de referencia del código base por defecto; solo copias de trabajo.
- Operaciones comunes (como commits, visualización del historial, y revertir cambios) son rápidas, porque no existe necesidad de comunicación con un servidor central.
- La comunicación con el servidor sólo es necesaria cuando se comparten cambios entre iguales (peers).
- Cada copia de trabajo funciona efectivamente como una copia de seguridad remota del codebase y de su historial de cambios, protegiéndolo de la pérdida de datos.

Otras diferencias incluyen:

- Múltiples repositorios “centrales”.
- El código discrepante de los repositorios es fusionado sobre la base de una red de confianza. Por ejemplo, el histórico de méritos o calidad de cambios.
- La red no está involucrada en operaciones comunes.
- Un conjunto separado de operaciones, de sincronización están disponibles para confirmar o recibir cambios con repositorios remotos.

Los propulsores del control de versiones distribuido señalan numerosas ventajas de los sistemas de control de versiones distribuido sobre el modelo centralizado tradicional:

- Permite a los usuarios trabajar de forma productiva cuando no están conectados a la red.
- Realiza las operaciones más rápido.
- Permite el trabajo en forma privada, de modo que los usuarios pueden utilizar sus cambios incluso en iteraciones intermedias que no quieren publicar.
- Evita confiar en una única máquina física como único punto de fallas.
- Permite el control centralizado de las versiones de lanzamiento del proyecto.

2) Explique en qué consisten los estados de Git, y de un ejemplo que ilustre los mismos.

R:

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en tu base de datos local. Modificado significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos. Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

El área de preparación es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice, pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

El flujo de trabajo básico en Git es algo así:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiendolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

3) Explique el término debugging y su importancia.

R:

Es el proceso rutinario de localización y eliminación de errores, anomalías o anomalías en los programas informáticos, que los programadores de software manejan metódicamente mediante herramientas de depuración. La depuración comprueba, detecta y corrige errores o errores para permitir el funcionamiento correcto del programa de acuerdo con las especificaciones establecidas.

Es un procedimiento importante en donde cualquier proceso nuevo de programación o mejora de equipos, se puede revisar en detalle para encontrar sus fallas y es muy útil para determinar el punto o la zona donde puedan estar fallando, puede parecer poco útil cuando se trata de un proyecto pequeño, pero es muy requerido y necesario cuando se tiene grandes cantidades de código.

4) Defina el término plug-in.

Un plugin es aquella aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. En nuestro idioma, por lo tanto, puede nombrarse al plugin como un complemento.

5) Defina JUnit 5 y su composición. Explique brevemente cada uno de sus componentes.

R:

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

Y sus componentes son:

JUnit Platform

JUnit Platform es la base que nos permite el lanzamiento de los frameworks de prueba en la JVMy, entre otras cosas, también es el encargado de proporcionarnos la posibilidad de lanzar la plataforma desde línea de comandos y de los plugins para Gradle y Maven.

JUnit Jupiter

JUnit Jupiter es el que más utilizaremos a la hora de programar. Nos permite utilizar el nuevo modelo de programación para la escritura de los nuevos tests de JUnit 5.

JUnit Vintage

Junit Vintage es el encargado de los tests de Junit 3 y 4, por si alguien los echa de menos.