TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Automatización de pruebas.
- Integración continua. Definición, objetivos e importancia.
- Herramientas de integración continua: Jenkins, Travis CI y Circle CI.
- Setup de herramientas con Jenkins.

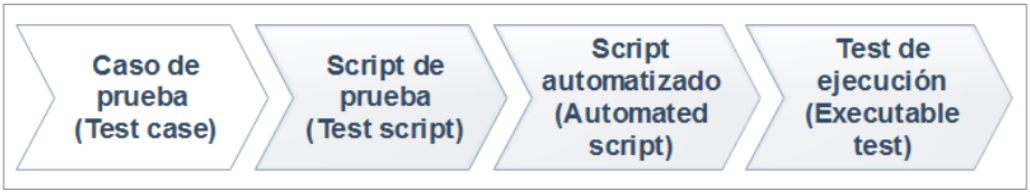
AUTOMATIZACIÓN DE PRUEBAS

En el mercado actual de las empresas de desarrollo de software, se necesita ser ágiles para mantenerse como competidores viables. Una de las claves para agilizar el proceso de desarrollo, es detectar errores antes, en etapas en las que nos cueste menos solucionarlos, lo que se logra testeando los sistemas, previo a ser instalados, en ambientes productivos. En el proceso de testing manual, podemos encontrar un concepto que hemos mencionado, pero que aún no habíamos definido, este es:

`Script` de prueba: representa el paso a paso que se debe ejecutar, de forma manual, para validar una condición del sistema.

Cuando se transforman estos scripts de prueba, en automatizados, utilizando herramientas especializadas, aparece el concepto de automatización de pruebas. Las pruebas automatizadas, tienen como objetivo detectar fallas en el software, evitando que una persona tenga que ejecutarlas manualmente, por lo cual no requieren una intervención en cada nueva aplicación. En este caso, el experto en testing, genera un caso a probar utilizando una herramienta, para que luego la misma se realice automáticamente, simulando la interacción humana con el software.

Cuando los scripts automatizados son utilizados en una iteración de un proceso de prueba, considerando, además, los datos que se utilizarán para la realización de éstas, aparecen en el término de test de ejecución.



La transformación de los `scripts`, que se visualiza en la imagen, es lo que se considera tradicionalmente como automatización en las pruebas de sistemas.
Se puede destacar, entre los beneficios de realizar automatización de pruebas, que permite cubrir un mayor porcentaje en un menor tiempo; reduce el esfuerzo en la ejecución de los test; permite entregar software de mayor calidad, en comparación con las pruebas manuales; asegura la mantenibilidad de los scripts de pruebas; se logra una disminución de horas de trabajo, asociadas a mantenciones correctivas y pruebas funcionales; facilitan la precisión a la hora de diagnosticar la falla detectada, y favorece los procesos de integración continua.

INTEGRACIÓN CONTINUA: DEFINICIÓN, OBJETIVOS E IMPORTANCIA

Es una práctica ágil, propuesta inicialmente en el marco de `eXtreme Programming`, a finales de los 90's, y pretendía servir como base para el cumplimiento del principio del manifiesto ágil, que se refiere a realizar entregas de software funcional frecuentemente. Vamos a revisar la definición propuesta por Martin Fowler en el año 2006:
Integración continua: es una práctica de desarrollo de software, en la cual los miembros de un equipo integran su trabajo frecuentemente, como mínimo de forma diaria. Cada integración se verifica mediante una herramienta de pruebas automática, para detectar los errores de integración tan pronto como sea posible.
Cada vez que un miembro del equipo realiza un `commit` en el control de versiones (git por ejemplo), el proyecto se compila, y se obtiene un ejecutable o `build` de manera automática, se desarrollan todas sus pruebas unitarias de forma automática, y se obtienen las métricas de calidad para detectar los errores tan pronto como sea posible. Esta práctica trae consigo múltiples ventajas, por ejemplo:

se facilita la corrección de errores; se alienta a los desarrolladores a realizar `merges` cada vez que completen tareas pequeñas; el equipo de desarrollo estará siempre trabajando con una versión actualizada del proyecto; cada `commit` será una posible versión; al mantener el repositorio al día de forma constante, y testear el proyecto por cada uno de los `commits` realizado en una rama concreta (comúnmente máster), evitaremos que en algún momento salgan a relucir cientos de conflictos testeables al realizar el `merge`; también favorece que se mantenga una calidad constante; incrementar la visibilidad del proceso; e incrementar la confianza entre los usuarios de negocio y el equipo de proyecto.

Prácticas de Integración Continua:

- Mantener un único repositorio de código fuente.
- Automatizar la construcción del proyecto.
- Hacer que la construcción del proyecto ejecute sus propios test.
- Entregar los cambios a la línea principal todos los días.
- Construir la línea principal en la máquina de integración.
- Mantener una ejecución rápida de la construcción del proyecto.
- Probar en una réplica del entorno de producción.
- Hacer que todo el mundo pueda obtener el último ejecutable de forma fácil.
- Publicar qué está pasando.
- Automatizar el despliegue.

HERRAMIENTAS DE INTEGRACIÓN CONTINUA

- **Jenkins**

Este software fue lanzado el año 2005 por la compañía Sun Mycrosistems, bajo el nombre de Hudson. Oracle compró la empresa, y unos años después, la comunidad de usuarios decidió cambiarle el nombre a `Jenkins`, y migrar el código a `Github` para continuar desarrollándolo constantemente. Escrito en Java, Jenkins es uno de los servidores de automatización de código abierto para la integración continua, con cientos de complementos, y una interfaz de usuario amigable. Se integra a la perfección con Git para ejecutar tareas con determinadas acciones `Git`, por ejemplo: cuando se lanza un `pull request`, cuando se hace un push a una determinada rama, cuando se crea una rama, entre otros. Además de integrarse con `Git`, también tiene la posibilidad de instalar muchos plugins para extender su funcionalidad. La base de `Jenkins` son las tareas, donde indicamos qué es lo que hay que hacer en un `build`. Ejemplo: podríamos programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, y cuando un desarrollador quiera subir su código al control de versiones, éste se compile y se ejecuten las pruebas.

Si el resultado no es el esperado, o hay algún error, `Jenkins` notificará al desarrollador, y al equipo de QA, por email o cualquier otro medio, para que lo solucione. Si el `build` es correcto, podremos indicar que intente integrar el código, y subirlo al repositorio de control de versiones.

Está disponible para Windows, Mac-OS, Linux y otras variantes de Unix. Proporciona procedimientos simples de instalación y actualización, y se puede configurar fácilmente a través de una interfaz gráfica de usuario. Está diseñado como un servidor de automatización extensible, por lo que puede usarse simplemente como un servidor de `CI`, o convertirse en un completo centro de entrega

continua. Ofrece notificaciones sobre el estado de la compilación, y admite la ejecución de comandos en los pasos previos a ésta, para permitir a los desarrolladores ajustar el flujo según sus preferencias.

En los aspectos negativos, podemos notar que la configuración de Jenkins se complicaría para los desarrolladores más principiantes o simplemente, para aquellos que no terminan de adaptarse a su uso. También hay desarrolladores que critican que su interfaz de usuario está poco actualizada, en comparación con otros competidores del mercado. Se recomienda su uso cuando se cuente con un servidor dedicado para el desarrollo y/o integración, ya que al instalar `Jenkins`, toda la información queda en una sola plataforma, ya sean los proyectos que se están desarrollando, los reportes generados, los estados de los `builds`, entre otros.

- **Travis CI.**

Es un sistema de integración continua programado en Ruby, multiplataforma, y gratuito para proyectos de código abierto. Está especialmente diseñado para construir y testear proyectos en `Github` de forma automática. No necesita instalación, se utiliza en línea. Para integrarlo, solo hay que iniciar sesión en `Travis`, darle acceso de lectura a nuestro código en el repositorio de interés, guardar en el directorio raíz del proyecto un sencillo archivo `YAML`, y definir los test necesarios. Cada vez que hagamos un `push` en nuestro repositorio, será notificado y comenzará con la ejecución del proceso.

Actualmente, soporta los sistemas Ubuntu y OS X, y los lenguajes y tecnologías Java, JavaScript, Node.js, iOS, PHP, Python, Ruby, C/C++, C#, Go, entre otros. Es la plataforma oficial de testeo para cada commit de Ruby, Rails, Rubinius, Rubygems, Bundler, Leiningen, Parrot o Symfony. `Travis CI` te permite conectar tu repositorio de `Github`, y probar después de cada push que hagas, regenerando el proyecto.

Una de sus ventajas más claras, es que el entorno de integración permite probar nuestras librerías o aplicaciones contra distintas configuraciones, sin necesidad de tenerlas instaladas localmente. Tienen varias máquinas virtuales preparadas para cada combinación,

y allí se puede instalar `MySQL`, o lo que necesitemos. La documentación está bien explicada y detalla cada proceso. También describe las herramientas y recursos de terceros. Además, se puede cambiar la configuración de la máquina virtual que se está utilizando, de ese modo podemos ejecutar nuestros test en un clon de la futura máquina de producción. También podemos configurar los pasos del `build` que se quieran (ejecución de test, generación de reportes, entre otros). Se recomienda su uso cuando no se cuenta con un servidor de desarrollo o de integración, para el desarrollo del software, ya que `Travis` brinda ese servicio.

Sin embargo, tiene algunas limitaciones, por ejemplo, solo es compatible con GitHub como controlador de versiones, y debe externalizar la muestra de reportes.

- **Circle CI.**

Fundada en 2011, con sede en San Francisco, y con una fuerza de trabajo remota global, libre de discriminaciones, `CircleCI` es una de las plataformas de integración y distribución continua (CI/CD) más grandes, y más utilizadas del mundo. Empresas como Spotify, Coinbase, Stitch Fix y BuzzFeed, utilizan esta solución para mejorar la productividad del equipo de ingeniería, lanzar mejores productos, y salir al mercado más rápido. Sus usuarios aprecian los paneles de control únicos con estadísticas y datos, sobre cómo trabajan los equipos y cómo se ejecuta su código, lo que les ayuda a mejorar y reducir el tiempo de comercialización.

Entre sus características, encontramos que, permite integrarse con distintas opciones de sistemas controladores de versiones, como: `GitHub`, `GitHub Enterprise` y `Bitbucket`. Además, al estar basada en la nube, es una herramienta flexible que se ejecuta, naturalmente, en cualquier entorno, como aplicaciones móviles multiplataforma, servidor Python API, o clúster Docker, con muchas opciones de personalización. Es compatible con diferentes lenguajes, incluidos Java, Python, JavaScript, Haskell, Ruby on Rails y Scala. Da la posibilidad

de seleccionar `Build Environment`. También, de cancelar automáticamente cualquier compilación en cola, o en ejecución, cuando se activa una más nueva. Divide y equilibra las pruebas en varios contenedores, para reducir el tiempo de construcción general.

Pone límites a los no administradores, impidiendo modificar la configuración crítica del proyecto. Tiene un nivel de uso gratuito, que provee varias funcionalidades, antes de tener que pagar.

SETUP DE HERRAMIENTAS CON JENKINS

Antes de comenzar a trabajar en un proyecto con integración continua, se deben revisar las herramientas que usaremos para lograrlo. Pensando en un equipo de desarrollo pequeño, necesitamos:

- Hardware: Preprocesador Intel I3 o similar mínimo, RAM 4GB.
- Espacio en disco libre: 50 GB mínimo.
- Sistemas operativos: Windows (64-bit), Linux, macOS, Ubuntu/Debian, entre otros.
- Jenkins: Version LTS (anual).
- Browser recomendado: Google Chrome, Mozilla Firefox, Apple Safari.
- Software: Java SE Development Kit 11.

`Jenkins` ofrece descargas para todo tipo de servidores y sistemas, por lo que no se van a presentar problema con ninguno.