Técnico en

DESARROLLO DE SOFTWARE

Diseño e Implementación del Software

Instituto Von Neumann

Unidad V

Diseño e Implementación del Software

<Implementación de Software>

Al momento de implementar algún tipo de software tenemos que tomar en cuenta la forma en que vamos a llevar el control de las diferentes versiones para evitar entorpecer la gestión de cambios, así como también para poder identificar qué cambios se incluyen en cada entrega, es por este motivo que existe la estrategia de git flows que nos ayuda a poder llevar a producción cambios de la mejor manera, así como también contribuye al manejo de diferentes features o funcionalidades que podemos llevar a producción en algún momento. En Git flows existen otras eventualidades que nos plantean soluciones tales como un fix o reparación de un error o un hotfix aplicado directamente a producción, indicando el flujo que deberíamos de seguir al momento de trabajar con alguna herramienta de versionamiento de código.

Git Flows

Git flows es un modelo alternativo el cual nos permite manejar un conjunto de ramas principales y ramas de funcionalidades en Git, las cuales nos permitirán integrar cambios hasta que estas se encuentren listas para implementarse. Comúnmente esta práctica es utilizada en situaciones donde se tienen programados los cambios a implementar, así como también en la práctica de DevOps de entrega continua.

Prácticamente Git flows nos aclara la forma en que tenemos que fusionar nuestras ramas y nos dicta cómo configurarlas.

Ramas principales y de desarrollo

En lugar de una única rama master, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama master o principal almacena el historial de publicación oficial y la rama de desarrollo o de desarrollo sirve como rama de integración para las funciones nuevas o correcciones que puedan existir.

Asimismo, conviene etiquetar todas las confirmaciones de la rama master con un número de versión.

El primer paso es complementar la rama master predeterminada con una rama develop. Una forma sencilla de hacerlo es que un desarrollador cree de forma local una rama develop vacía y la envíe al servidor.

Ramas feature/Característica

Toda rama de una nueva característica o funcionalidad a implementar al sistema debe de tener su propia rama a partir de la rama de desarrollo.

Esto permitirá que al finalizar dicha funcionalidad los cambios se integren a la rama de desarrollo, tomando en cuenta siempre que en ningún momento estos cambios deben de interactuar directamente con la rama principal o master.

Ramas de publicación (RELEASES)

Una rama de publicacion se genera a partir de una rama de desarrollo en la cual ya se encuentren acumulados los cambios necesarios para su publicación permitiendo generar una rama con dicha version a implementar en un ambiente productivo.

Al momento de poder fusionar una rama de publicacion con una rama master se debe de generar una etiqueta donde generalice los cambios integrados dentro de cada uno de los distintos elementos que se van a desplegar.

Utilizar una rama específica para preparar publicaciones hace posible que un equipo perfeccione la publicación actual mientras otro equipo sigue trabajando en las funciones para la siguiente publicación. Asimismo, crea fases de desarrollo bien definidas (por ejemplo, es fácil decir: "Esta semana nos estamos preparando para la versión 4.0" y verlo escrito en la estructura del repositorio).

Ramas de hotfix/corrección

Las ramas de mantenimiento, de corrección o de hotfix sirven para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la rama master y no en la develop. Esta es la única rama que debería bifurcarse directamente a partir de master. Cuando se haya terminado de aplicar la corrección, debería fusionarse en master y develop (o la rama release actual), y main debería etiquetarse con un número de versión actualizado.

Tener una línea de desarrollo específica para la corrección de errores permite que tu equipo aborde las incidencias sin interrumpir el resto del flujo de trabajo ni esperar al siguiente ciclo de publicación. Puedes concebir las ramas de mantenimiento como ramas release ad hoc que trabajan directamente con la rama master.

<Pruebas de software>

La calidad del software es una característica que debemos de tomar en cuenta al momento de trabajar cualquier tipo de proyecto, es algo de lo cual no debe de prescindir ningún software ya que debemos de otorgar siempre al cliente un sistema estable y con la cantidad mínima de fallas posibles.

Algunas de las pruebas que podemos realizar al momento de trabajar con proyectos de software son:

Pruebas funcionales

Este tipo de pruebas como su nombre indica validan la funcionalidad de los distintos componentes desarrollados de diferentes formas, tomando en cuenta los siguientes tipos:

1. **Pruebas Unitarias**

   Este tipo de pruebas es utilizado para poder verificar por parte del desarrollador cada una de las porciones de código realizadas cumpla con lo que debe de hacer, este tipo de pruebas es de mucha ayuda cuando realizamos alguna modificación a nivel de código ya que nos permite el poder detectar si la prueba estaba mal diseñada o simplemente modificamos algo que no se cumple al momento de ejecutarlas.

2. **Prueba de componentes**

   En la prueba de componentes es necesario validar que dicho componente funcione de la forma deseada, identificando las entradas, procesos y salida de información para garantizar su buen funcionamiento. Por ejemplo:

- Prueba de UI, para usabilidad y accesibilidad
- Prueba de carga, para asegurar el rendimiento
- Inyección de SQL a través de componentes de UI para asegurar la seguridad
- Prueba de login, con credenciales válidas e inválidas

3. Prueba de humo

Este tipo de pruebas validan únicamente que las funcionalidades más importantes del software se ejecuten de forma correcta utilizando pruebas sencillas y rápidas para asegurar su funcionamiento.

Las pruebas de humos prácticamente serían una las primeras que debemos de tomar en cuenta al momento de modificar alguna aplicación ya que nos dirán de forma inmediata si algo anda mal o todo está bien de lo que vamos a validar. Después de validar que todo está correctamente se puede seguir en las validaciones de pruebas funcionales o de regresión.

4. Pruebas de integración

Las pruebas de integración son muy populares dentro de las pruebas funcionales, ya que tomamos en cuenta el comportamiento y funcionamiento de todos los componentes que deben de interactuar entre sí ya que el desarrollador comúnmente valida las funcionalidades únicamente del componente que modifica. Este tipo de prueba nos permite tomar el sistema conformado por múltiples componentes como uno solo y verificar que cumplan el objetivo del porque interactúan entre sí.

5. Pruebas de regresión

Las pruebas de regresión son muy útiles al momento de realizar por parte del desarrollador algún tipo de cambio al sistema, nos permiten el poder asegurarnos de que las funcionalidades previas a la modificación siguen funcionando sin ningún problema y no se vieron afectadas por los nuevos cambios.

6. **Prueba de cordura**

   La prueba de cordura comúnmente se ejecuta cuando se realizan cambios menores a nivel de código, esto nos permite verificar si las correcciones de algún tipo de error se ejecutaron de forma correcta y hayan solucionado corregir el problema de forma efectiva.

7. **Pruebas de aceptación del usuario**

   Las pruebas de aceptación del usuario comúnmente es lo último que podemos ejecutar para validar que los requerimientos del usuario se cumplan de forma correcta. Este tipo de pruebas se debería de ejecutar dentro de un ambiente parecido al entorno de producción, en algunas ocasiones este tipo de prueba se puede realizar al momento de entregar algún sistema o producto.

Pruebas no funcionales

Este tipo de pruebas a diferencia de las funcionales nos permiten identificar riesgos que puedan afectar el funcionamiento óptimo de nuestras aplicaciones. Las pruebas no funcionales se realizan con el fin de obtener información o métricas para poder identificar el rendimiento de la aplicación, así como también si el producto cumple las expectativas del cliente.

Algunos tipos de pruebas no funcionales son:

1. **Pruebas de carga**

La prueba de carga se realiza con el objetivo de poder obtener la información de cómo funciona el sistema cuando existe cierta cantidad de usuarios dentro del sistema, validando la respuesta que esta lanza al momento de cualquier solicitud que se realice.

Ejemplo: El cliente nos solicita que el producto pueda soportar la carga de 1000 usuarios de forma simultánea. Este resultado se compara con el volumen esperado.

2. **Pruebas de rendimiento**

   Las pruebas de rendimiento como su nombre indica es poder identificar el funcionamiento que tiene el sistema con diferentes tipos de carga controlados.

   Ejemplo: Se debe de estimar cual es el comportamiento que debe de tener el sistema al enviar 50,500 y 5000 peticiones y poder comparar el resultado del rendimiento de la aplicación en todos los escenarios.

3. **Pruebas de estrés**

   Este tipo de pruebas no fucionales nos permiten el poder encontrar los puntos de quiebre del sistema, es decir, cuantos usuarios, peticiones o tiempos que se soportan dentro de la aplicación. Este tipo de pruebas es parecida a las pruebas de rendimiento y de carga pero se diferencian en que una prueba de estrés debe de superar la cantidad esperada o solicitada por el cliente realizando dicha prueba hasta que el sistema deje de responder.

<Método de procedimiento (MOP)>

Un método de procedimiento no es más que un documento generado mediante el cual se indica el procedimiento a ejecutar para poder llevar a cabo la instalación de un sistema, cambio o aplicación a nivel técnico de forma correcta. Este tipo de documentos es importante para la mayoría de las empresas que lleva un registro de los cambios a realizar para poder contar con un respaldo ante cualquier eventualidad y poder identificar la forma en que se pueda actuar, así como también controlar la cantidad de cambios que se han realizado y en qué plataformas.

Las partes que se deben de incluir un MOP son:

- Control de versiones o cambios del documento.
- Encargado técnico, quien es el que gestiona el cambio.
- Descripción/Justificación del cambio.
- Pasos para llevar a cabo para su correcta instalación en ambiente productivo (Rollout).
- Validaciones para realizar en caso sean exitosos los cambios.
- Pasos para llevar a cabo para su correcto rollback en caso de que las validaciones no sean positivas.
- Persona que ejecutará el cambio.

En algunas ocasiones, es importante en las empresas el poder mantener un equipo dedicado en atender cualquier incidencia que pueda ocurrir a causa de las nuevas implementaciones, este equipo prácticamente será la niñera que atenderá cualquier inconveniente que pueda surgir en el sistema, así como también ejecutar el rollback en caso de fallas que no se puedan recuperar.

Referencias de contenido

- https://www.atlassian.com/git/tutorials/why-git
- https://www.atlassian.com/git/tutorials/what-is-git
- http://www.lnds.net/blog/2010/07/control-de-versiones-distribuido.html
- https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones
- https://azure.microsoft.com/es-es/overview/what-is-a-virtual-machine/
- https://www.1and1.es/digitalguide/servidores/know-how/desarrollo-web-con-stacks-de-software/
- http://www.pmoinformatica.com/2012/09/ambientes-de-desarrollo-de-software.html