

Ingeniería del software y metodologías ágiles

Rodrigo Corral

rcorral@plainconcepts.com

<http://geeks.ms/blogs/rcorral>

MVP Team System / CSM / CSP

Plain Concepts

Ingeniería del software



SOCORRO !



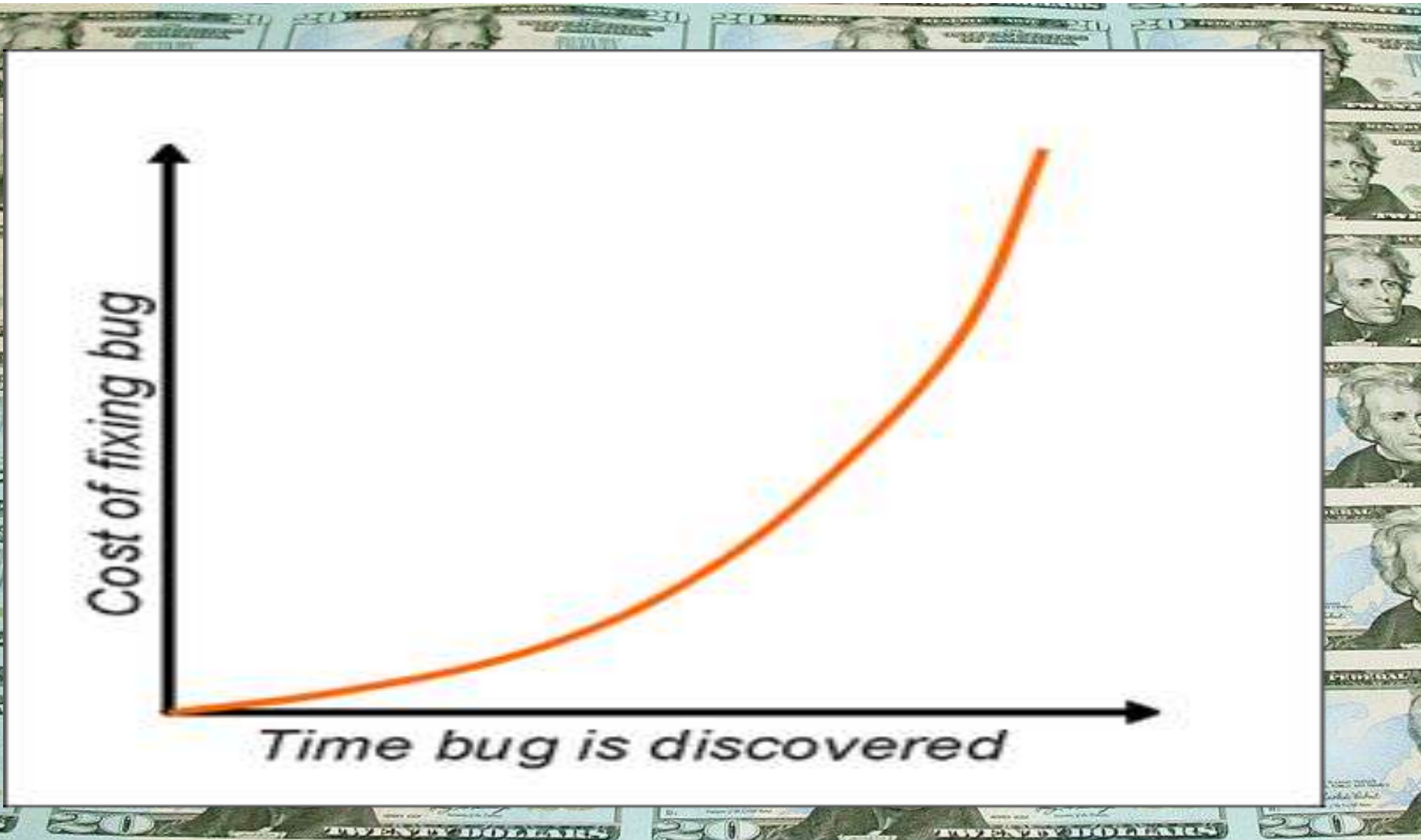
Pruebas unitarias

Gestión de la configuración

Integración continua

Más en próximos capítulos... ;)

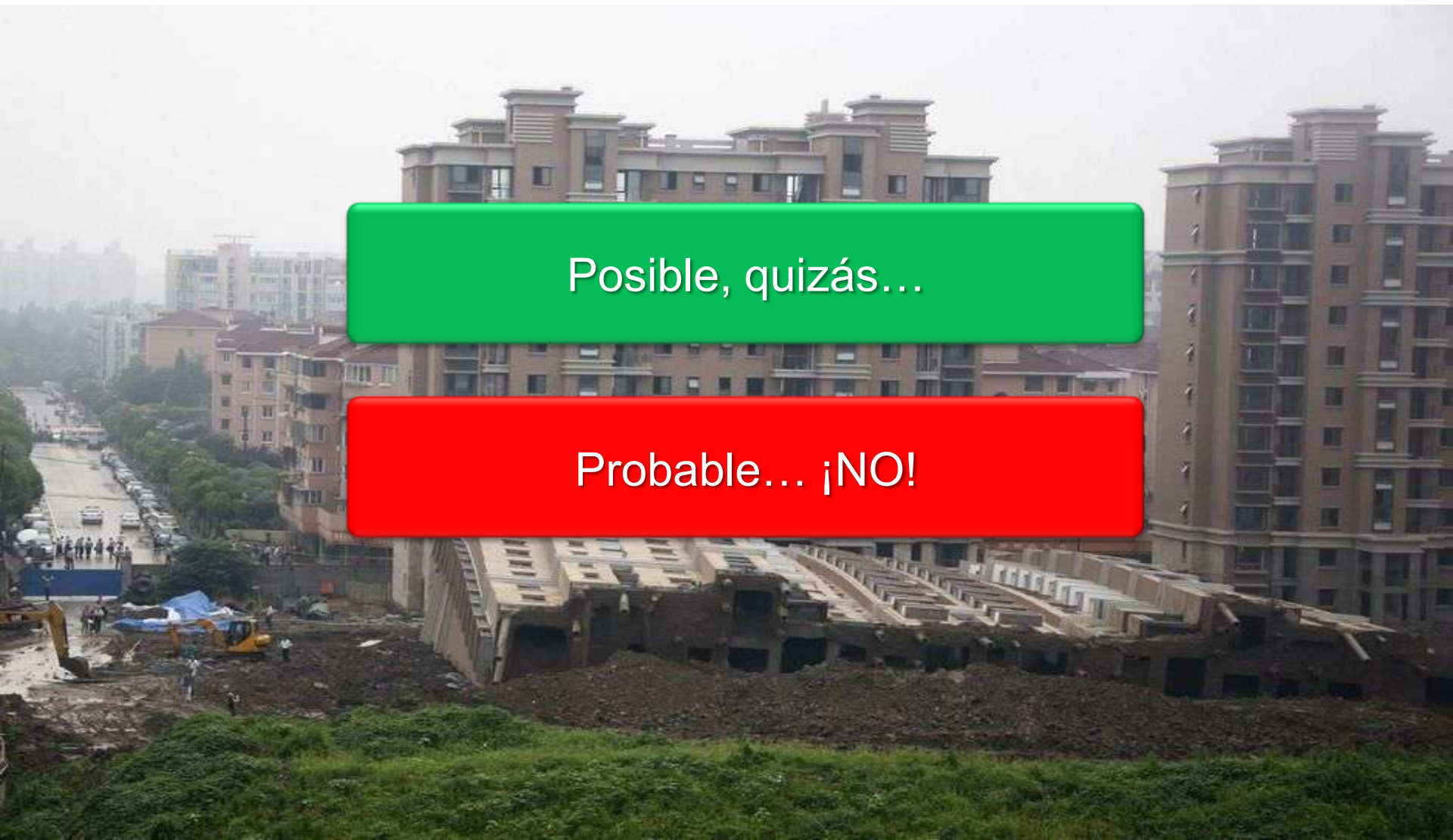
¿Qué es la ingeniería del software?



¿Es posible la agilidad sin buenos fundamentos de ingeniería del software?

Posible, quizás...

Probable... ¡NO!



Pruebas unitarias

- La detección más temprana posible
- Demostración de que no hemos roto nada
- Documentación
- Marcador claro de que una tarea está completada
- Mejora el diseño
- Verifica la correcta corrección de errores
- El tiempo de depuración se reduce

Pruebas unitarias

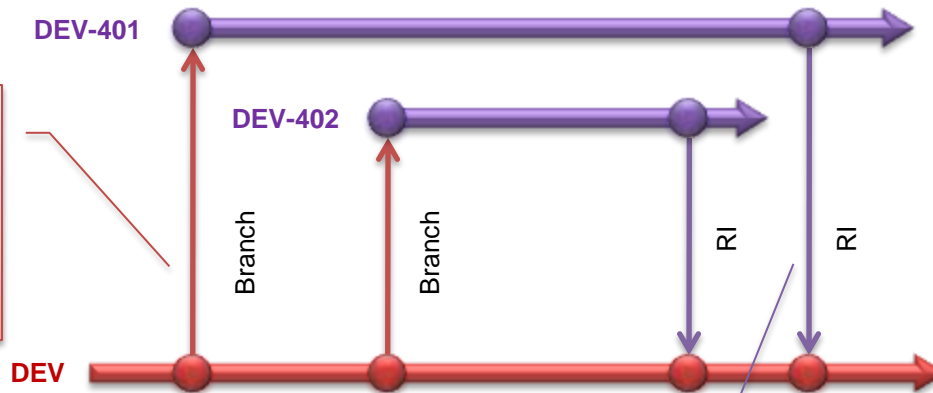
- ¿Cómo son las buenas pruebas unitarias?
 - Se ejecuta rápido, se ejecuta rápido, se ejecuta rápido
 - Tiene escasas dependencias
 - Su alcance es claro y limitado
 - Se ejecutan y pasan de manera independiente.
 - Revelan claramente su intención
 - Tienen la mayor cobertura posible
 - No alteran el estado del sistema

Gestión de la configuración

- Desarrollo concurrente y en equipo
- 'Aislar' el entorno de pruebas
- Lograr incrementos de funcionalidad potencialmente entregables
- Habilitar mecanismos ágiles y operativos para la corrección de errores

Desarrollo concurrente y en equipo

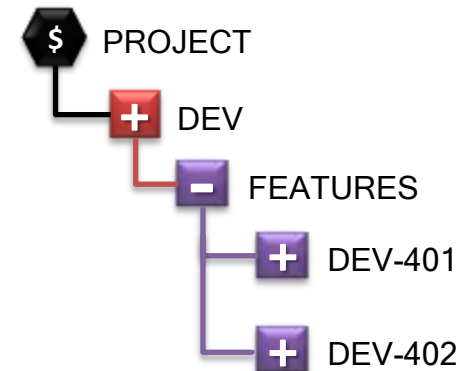
Estructura de ramas



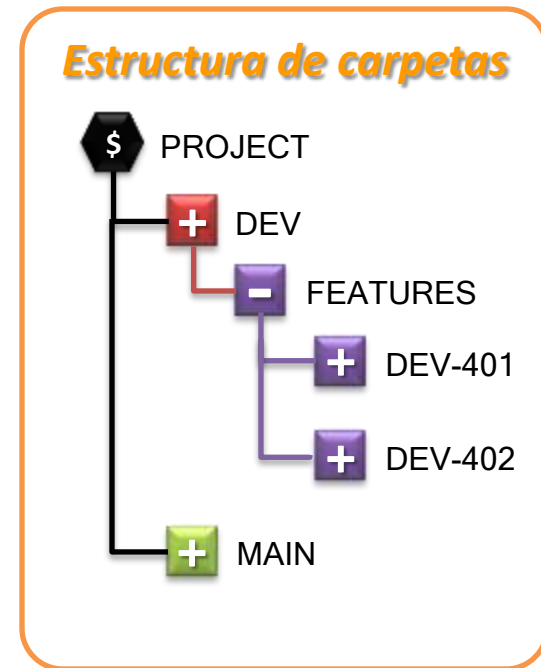
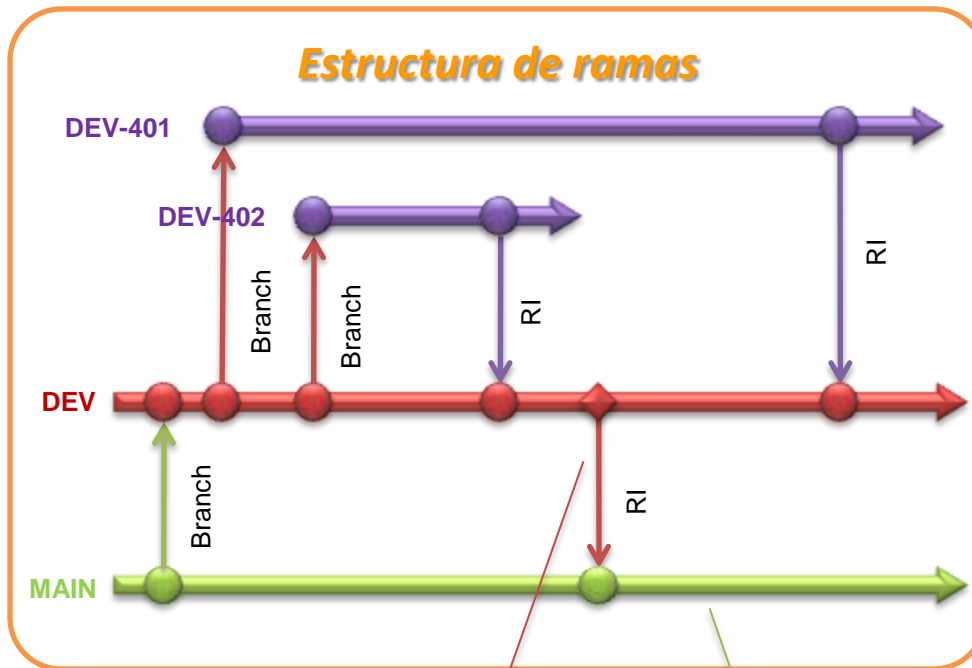
Antes de comenzar a trabajar en una historia de usuario creamos una rama sobre la que realizamos el desarrollo

Concluido el desarrollo de la historia de usuario, integramos el código en la rama principal de desarrollo

Estructura de carpetas



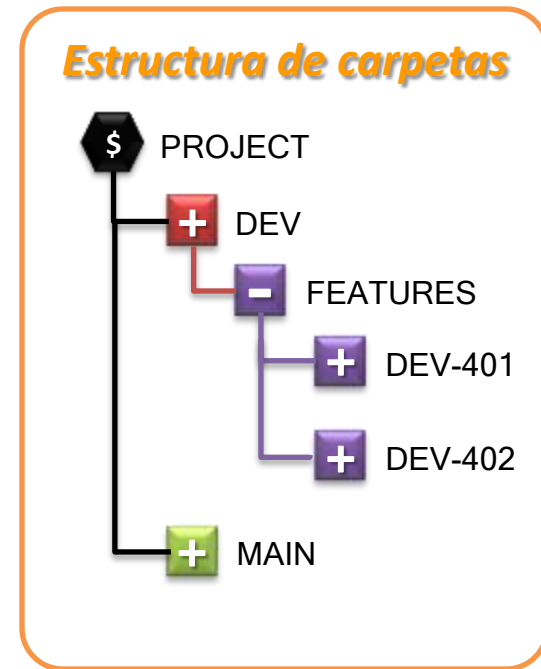
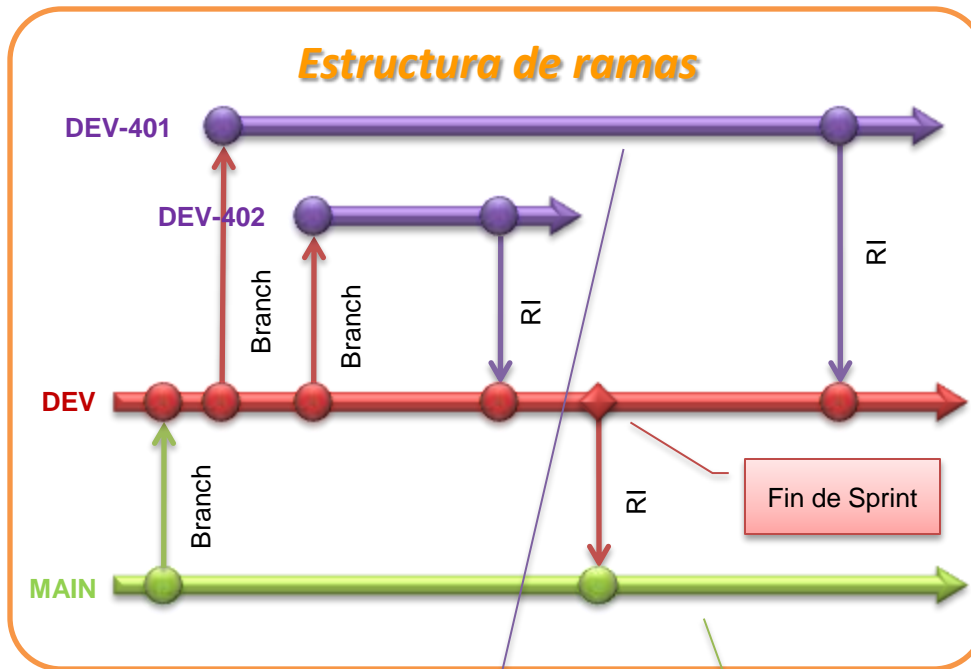
'Aislar' el entorno de pruebas



Quando se cumplen las condiciones de calidad el código en desarrollo se integra en la rama MAIN para que los testers comiencen el trabajo de estabilización

Quando se cumplen las condiciones de calidad el código en desarrollo se integra en la rama MAIN para que los testers comiencen el trabajo de estabilización

Incrementos de funcionalidad potencialmente entregables

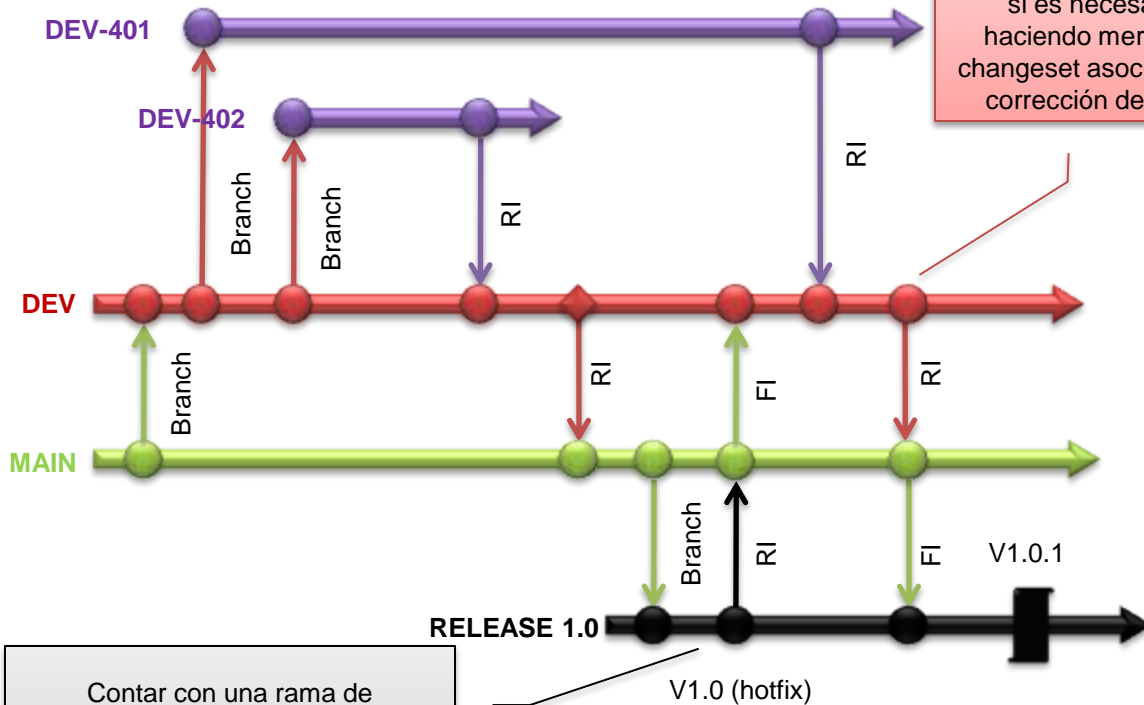


Realizar el desarrollo de nuevas funcionalidades sobre ramas dedicadas permite que si una funcionalidad no se completa a tiempo para incluirla en el Sprint Review el resto de la base de código principal siga siendo coherente y no incluya características incompletas

Usar ramas de característica garantiza que a la rama principal, sobre la que realizamos la estabilización del software, solo contendrá características completas y que han alcanzado un mínimo de calidad que permita que el trabajo de los testers sea productivo

Corrección de errores

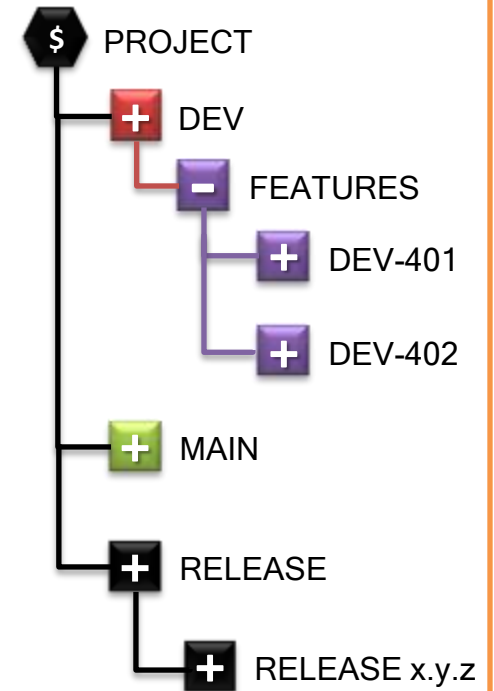
Estructura de ramas



Los errores que no son urgentes se corrigen sobre la línea principal de desarrollo y se llevan a las ramas de release, si es necesario, haciendo merge del changeset asociado a la corrección del error

Contar con una rama de RELEASE nos permite liberar parches de emergencia, para 'showstoppers', minimizando los posibles impactos sobre el entorno de producción y minimizando las necesidades de validación por parte de los testers

Estructura de carpetas



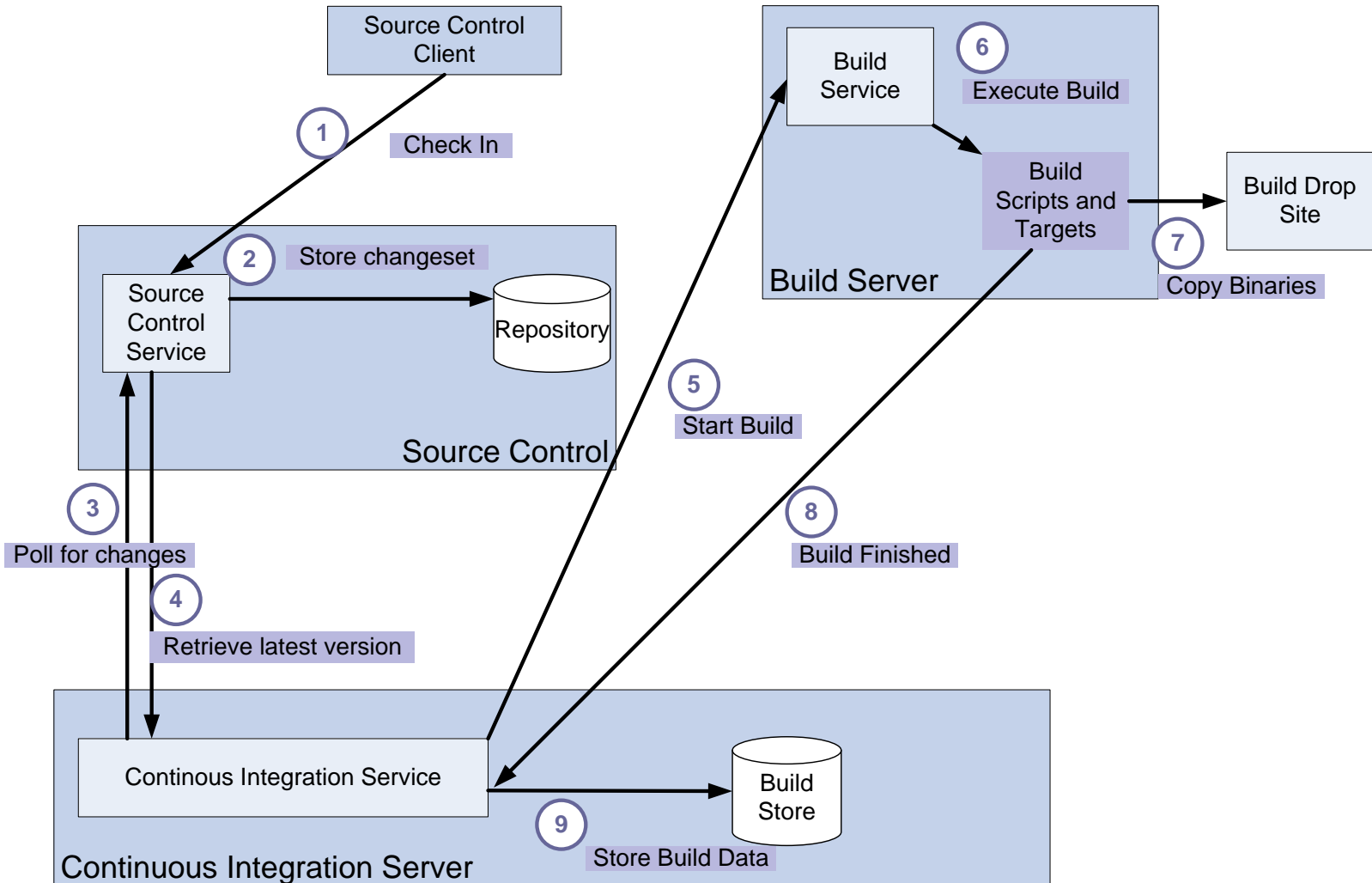
Construcción automatizada

- Construcción automatizada: patrón 'one command complete build'
- ¿Qué es completo?
 - Define tu nivel de completo para tu 'build'.
- Automatiza, automatiza, automatiza.
- Se puede compilar un kernel luego se puede compilar automáticamente tu proyecto.
- ¡No hay excusas!
- Entorno neutral
 - En mi máquina compila.
 - En mi máquina pasan los test.
 - En mi máquina funciona.

Integración continua

- Precondición: construcción automatizada.
- Detección más temprana posible de:
 - Errores de integración.
 - Regresiones en las pruebas unitarias.
- Evita que la ejecución de los test unitarios se supedite a la voluntad del desarrollador.
- Debe proteger todas las ramas donde se integra código.
- Ocurre en cada check-in.
- Si se rompe una build la prioridad es corregirla.

Integración continua



Integración continua

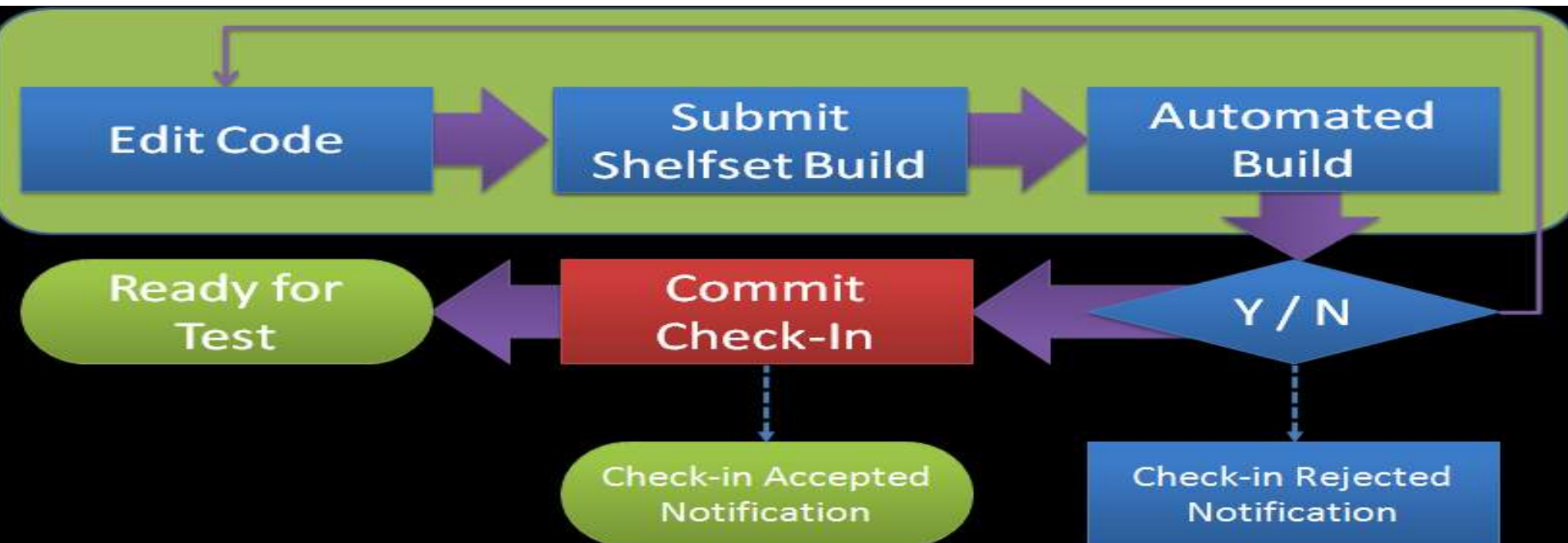
- Ventajas:
 - Los problemas de integración se detectan y corrigen continuamente.
 - Alerta temprana de código erróneo/incompatible.
 - Test unitario inmediato de todos los cambios.
 - Disponibilidad total de una build “actual” para una demo de pruebas o para ser liberada.
 - El impacto inmediato del check in de código erróneo actúa como incentivo para no meter la pata.

Integración continua

- Inversiones:
 - Necesita mucho mantenimiento.
 - Necesita servidores de compilación.
 - El impacto de los fallos actúa como incentivo negativo para hacer check-ins frecuentes (backup check-in).
 - **El código erróneo solo se detecta una vez añadido al repositorio.**

Gated check-ins

- Cuando se rompe la build
 - Es tarde el código ya está integrado
- Todo el equipo se ve afectado
- Alternativa: gated check-ins.



¿Preguntas?