Universidad Nacional Arturo Jauretche
Ingeniería de Software I
Testing e Integración
Dr. Sergio D. Conde

MATERIA: Ingeniería de Software
LECTURA: Testing e Integración

Validando las Pruebas

Las pruebas no pueden demostrar que el software está libre de defectos o que se comportará en todo momento como está especificado. Siempre es posible que una prueba que se haya pasado por alto pueda descubrir problemas adicionales con el sistema.

Generalmente, por lo tanto, el objetivo de las pruebas del software es convencer a los desarrolladores del sistema y a los clientes de que el software es lo suficientemente bueno para su uso operacional. La prueba es un proceso que intenta proporcionar confianza en el software.

Los casos de prueba son especificaciones de las entradas para la prueba y la salida esperada del sistema más una afirmación de lo que se está probando. Los datos de prueba son las entradas que han sido ideadas para probar el sistema. Los datos de prueba a veces pueden generarse automáticamente.

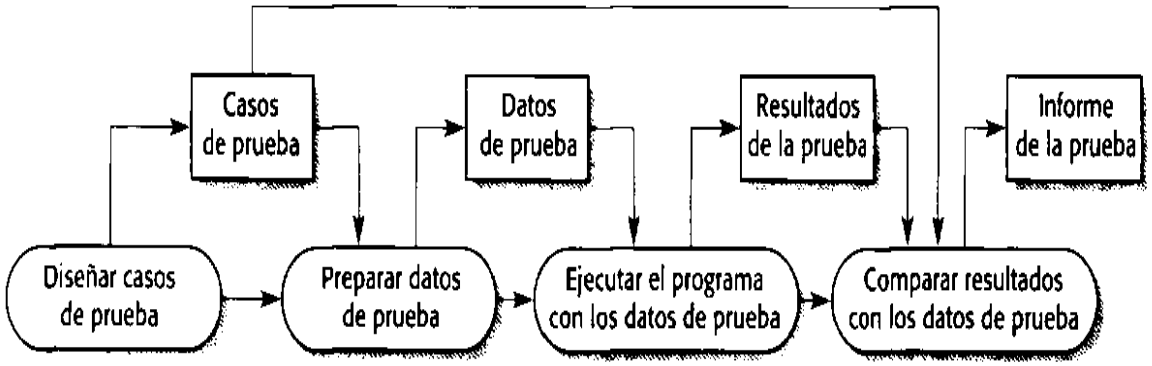
La generación automática de casos de prueba es imposible. Las salidas de las pruebas sólo pueden predecirse por personas que comprenden lo que debería hacer el sistema.

Las pruebas exhaustivas, en las que cada posible secuencia de ejecución del programa es probada, son imposibles. Las pruebas, por lo tanto, tienen que basarse en un subconjunto de posibles casos de prueba. Idealmente, algunas compañías deberían tener políticas para elegir este subconjunto en lugar de dejar esto al equipo de desarrollo. Estas políticas podrían basarse en políticas generales de pruebas, tal como una política en la que todas las sentencias de los programas deberían ejecutarse al menos una vez. De forma alternativa, las políticas de pruebas pueden basarse en la experiencia de uso del sistema y pueden centrarse en probar las características del sistema operacional. Por ejemplo:

1. Deberían probarse todas las funciones del sistema a las que se accede a través de menús.

2. Deben probarse todas las combinaciones de funciones (por ejemplo, formateado de textos) a las que se accede a través del mismo menú.

3. En los puntos del programa en los que el usuario introduce datos, todas las funciones deben probarse con datos correctos e incorrectos.



Un modelo del proceso de pruebas del software.

1. **Pruebas del sistema**

Para la mayoría de los sistemas complejos, existen dos fases distintas de pruebas del sistema:

1.1. ***Pruebas de integración,*** en las que el equipo de pruebas tiene acceso al código fuente del sistema. Cuando se descubre un problema, el equipo de integración intenta encontrar la fuente del problema e identificar los componentes que tienen que ser depurados.

Las pruebas de integración se ocupan principalmente de encontrar defectos en el sistema.

1.2. ***Pruebas de entregas,*** en las que se prueba una versión del sistema que podría ser entregada a los usuarios. Aquí, el equipo de pruebas se ocupa de validar que el sistema satisface sus requerimientos y con asegurar que el sistema es confiable.

Universidad Nacional Arturo Jauretche
Ingeniería de Software I
Testing e Integración
Dr. Sergio D. Conde

Las pruebas de entregas son normalmente pruebas de «caja negra» en las que el equipo de pruebas se ocupa simplemente de demostrar si el sistema funciona o no correctamente. Los problemas son comunicados al equipo de desarrollo cuyo trabajo es depurar el programa.

Cuando los clientes se implican en las pruebas de entregas, éstas a menudo se denominan ***pruebas de aceptación.*** Si la entrega es lo suficientemente buena, el cliente puede entonces aceptarla para su uso.

Fundamentalmente, se puede pensar en las pruebas de integración como las pruebas de sistemas incompletos compuestos por grupos de componentes del sistema.

Las pruebas de entregas consisten en probar la entrega del sistema que se pretende proporcionar a los clientes.

Naturalmente, éstas se solapan, en especial cuando se utiliza desarrollo incrementa! y el sistema para entregar está incompleto. Generalmente, la prioridad en las pruebas de integración es descubrir defectos en el sistema, y la prioridad en las pruebas del sistema es validar que el sistema satisface sus requerimientos. Sin embargo, en la práctica, hay una parte de prueba de validación y una parte de prueba de defectos durante ambos procesos

1.1. Pruebas de integración

La integración del sistema implica identificar grupos de componentes que proporcionan alguna funcionalidad del sistema e integrar éstos añadiendo código para hacer que funcionen conjuntamente.

Algunas veces, primero se desarrolla el esqueleto del sistema en su totalidad, y se le añaden los componentes. Esto se denomina ***integración descendente.***

De forma alternativa, pueden integrarse primero los componentes de infraestructura que proporcionan servicios comunes, tales como **el acceso a bases de datos y redes,** y a continuación pueden añadirse los componentes funcionales.

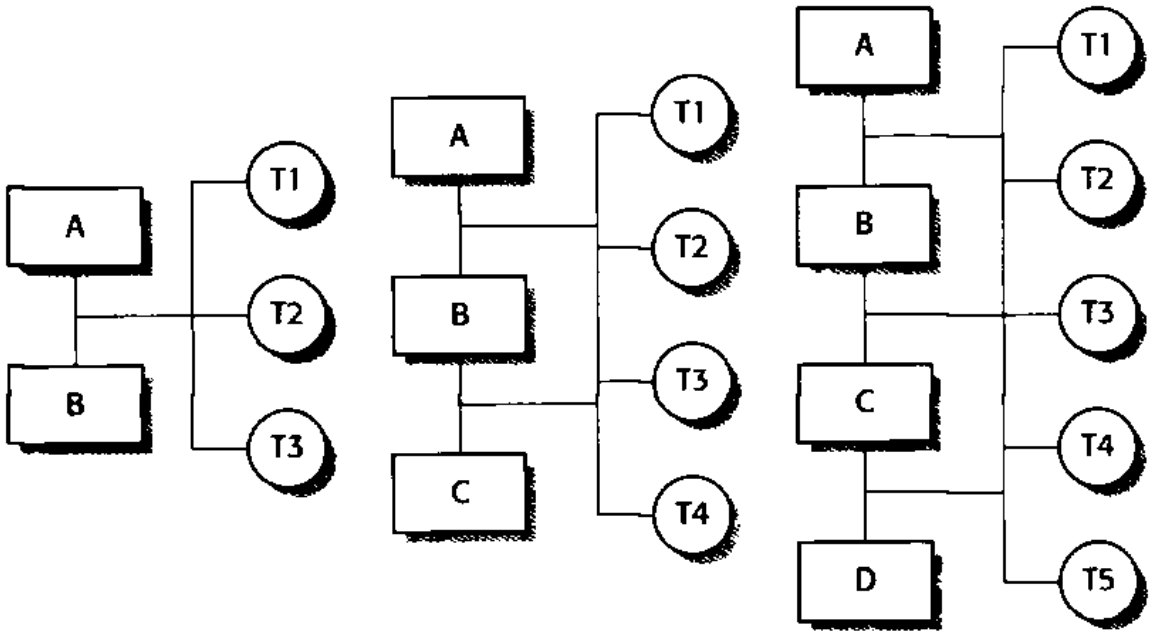
Ésta es la ***integración ascendente.***

En la práctica, para muchos sistemas, la estrategia de integración es una mezcla de ambas, añadiendo en incrementos componentes de infraestructura y componentes funcionales. En ambas aproximaciones de integración, normalmente tiene que desarrollarse código adicional para simular otros componentes y permitir que el sistema se ejecute.

La principal dificultad que surge durante las pruebas de integración es la localización de los errores. Existen interacciones complejas entre los componentes del sistema, y cuando se descubre una salida anómala, puede resultar difícil identificar dónde ha ocurrido el error.

Para hacer más fácil la localización de errores, siempre debería utilizarse una aproximación incremental para la integración y pruebas del sistema. Inicialmente, debería integrarse una configuración del sistema mínima y probar este sistema.

Cuando se planifica la integración, tiene que decidirse el orden de integración de los componentes.



1.1.1. Pruebas de integración increméntales.

A, B, C y D son componentes, y desde TI hasta T5 son conjuntos de pruebas relacionados de las características incorporadas al sistema. T I , T2 y T3 se ejecutan primero sobre un sistema formado por los componentes A y B (el sistema mínimo).

Si tales componentes revelan defectos, éstos se corrigen. El componente C se integra y TI, T2 y T3 se repiten para asegurar que no ha habido interacciones no esperadas con A y B. Si surgen problemas en estas pruebas, esto significa probablemente que son debidos a las interacciones con el nuevo componente. Se localiza el origen del problema, simplificando así la localización y reparación de defectos. El conjunto de pruebas T4 se ejecuta también sobre el sistema. Finalmente, el componente D se integra y se prueba utilizando las pruebas existentes y nuevas (T5).

1.2. Pruebas de entregas

Las pruebas de entregas son el proceso de probar una entrega del sistema que será distribuida a los clientes. El principal objetivo de este proceso es incrementar la confianza del suministrador en que el sistema satisface sus requerimientos. Si es así, éste puede entregarse como un producto o ser entregado al cliente. Para demostrar que el sistema satisface sus requerimientos, tiene que mostrarse que éste entrega la funcionalidad especificada, rendimiento y confiabilidad, y que no falla durante su uso normal.

Las pruebas de entregas son normalmente un proceso de pruebas de caja negra en las que las pruebas se derivan a partir de la especificación del sistema. El sistema se trata como una caja negra cuyo comportamiento sólo puede ser determinado estudiando sus entradas y sus salidas relacionadas.

Otro nombre para esto es ***pruebas funcionales,*** debido a que al probador sólo le interesa la funcionalidad y no la implementación del software.

Entrada de Datos ……..> Sistema …….> Resultados de la Prueba

2. Pruebas de rendimiento

Una vez que un sistema se ha integrado completamente, es posible probar las propiedades emergentes del sistema tales como rendimiento y fiabilidad. Las pruebas de rendimiento tienen que diseñarse para asegurar que el sistema pueda procesar su carga esperada. (Por ejemplo: debe procesar 1000 operaciones en dos horas).

Esto normalmente implica planificar una serie de pruebas en las que la carga se va incrementando regularmente hasta que el rendimiento del sistema se hace inaceptable.

Este tipo de pruebas tienen dos funciones:

1. Prueba el comportamiento de fallo de ejecución del sistema. Pueden aparecer circunstancias a través de una combinación no esperada de eventos en donde la carga sobre el sistema supere la máxima carga anticipada. En estas circunstancias, es importante que un fallo de ejecución del sistema no provoque la corrupción de los datos o pérdidas inesperadas de servicios de los usuarios. Las pruebas de estrés verifican que las sobrecargas en el sistema provocan «fallos ligeros» en lugar de colapsarlo bajo su carga.
2. Sobrecargan el sistema y pueden provocar que se manifiesten defectos que normalmente no serían descubiertos. Aunque se puede argumentar que estos defectos es improbable que causen fallos de funcionamiento en un uso normal, puede haber combinaciones inusuales de circunstancias normales que las pruebas de estrés pueden reproducir.

3. Pruebas de componentes

Las pruebas de componentes (a menudo denominadas pruebas de unidad) son el proceso de probar los componentes individuales en el sistema. Éste es un proceso de pruebas de defectos, por lo que su objetivo es encontrar defectos en estos componentes. Tal y como se indicó en la introducción, para la mayoría de los sistemas, los desarrolladores de componentes son los responsables de las pruebas de componentes.

Existen diferentes tipos de componentes que pueden probarse en esta etapa:

1. Funciones individuales o métodos dentro de un objeto.

2. Clases de objetos que tienen varios atributos y métodos.

3. Componentes compuestos formados por diferentes objetos o funciones. Estos componentes compuestos tienen una interfaz definida que se utiliza para acceder a su funcionalidad.

Las funciones o métodos individuales son el tipo más simple de componente y sus pruebas son un conjunto de llamadas u estas rutinas con diferentes parámetros de entrada. Pueden utilizarse las aproximaciones para diseñar los casos de prueba, descritos en la sección siguiente, y para diseñar las pruebas de las funciones o métodos.

Cuando se están probando clases de objetos, deberían diseñar las pruebas para proporcionar cobertura para todas las características del objeto. Por lo tanto, las pruebas de clases de objetos deberían incluir:

1. Las pruebas aisladas de todas las operaciones asociadas con el objeto.

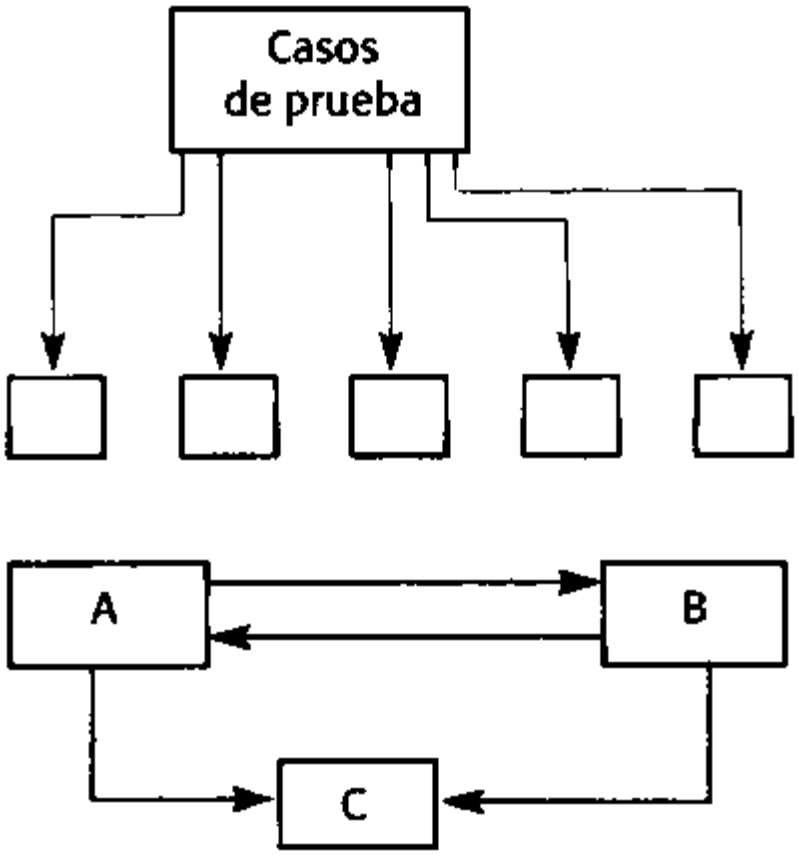
2. La asignación y consulta de todos los atributos asociados con el objeto.

3. Ejecutar el objeto en todos sus posibles estados. Esto significa que deben simularse todos los eventos que provocan un cambio de estado en el objeto.

Universidad Nacional Arturo Jauretche
Ingeniería de Software I
Testing e Integración
Dr. Sergio D. Conde

4. Pruebas de interfaces

Muchos componentes en un sistema no son simples funciones u objetos, sino que son componentes compuestos formados por varios objetos que interactúan.



Supongamos que los componentes A, B y C se han integrado para formar un componente más grande o subsistema. Los casos de prueba no se aplican a componentes individuales, sino a la interfaz del componente compuesto que se ha creado combinando estos componentes.

Las pruebas de interfaces son particularmente importantes para el desarrollo orientado a objetos y basado en componentes. Los objetos y componentes se definen por sus interfaces y pueden ser reutilizados en combinación con otros componentes en sistemas diferentes.

Los errores de interfaz en el componente compuesto no pueden detectarse probando los objetos individuales o componentes. Los errores en el componente compuesto pueden surgir debido a interacciones entre sus partes.

Existen diferentes tipos de interfaces entre los componentes del programa y consecuentemente, distintos **tipos de errores de interfaces** que pueden producirse:

1. ***Interfaces de parámetros.*** Son interfaces en las que los datos, o algunas veces referencias a funciones, se pasan de un componente a otro en forma de parámetros.
2. ***Interfaces de memoria compartida.*** Son interfaces en las que un bloque de memoria se comparte entre los componentes. Los datos se colocan en la memoria por un subsistema y son recuperados desde aquí por otros subsistemas.
3. ***Interfaces procedentes.*** Son interfaces en las que un componente encapsula un conjunto de procedimientos que pueden ser llamados por otros componentes. Los objetos y los componentes reutilizables tienen esta forma de interfaz.
4. ***Interfaces de paso de mensajes.*** Son interfaces en las que un componente solicita un servicio de otro componente mediante el paso de un mensaje. Un mensaje de retorno incluye los resultados de la ejecución del servicio. Algunos sistemas orientados a objetos tienen esta forma de interfaz, así como los sistemas cliente-servidor.

Los errores de interfaces son una de las formas más comunes de error en sistemas complejos. Estos errores se clasifican en tres clases:

1. ***Mal uso de la interfaz.*** Un componente llama a otro componente y comete un error en la utilización de su interfaz. Este tipo de errores es particularmente común con interfaces de parámetros en donde los parámetros pueden ser de tipo erróneo, pueden pasarse en el orden equivocado o puede pasarse un número erróneo de parámetros.
2. ***No comprensión de la interfaz.*** El componente que realiza la llamada no comprende la especificación de la interfaz del componente al que llama, y hace suposiciones sobre el comportamiento del componente invocado. El componente invocado no se comporta como era de esperar y esto provoca un comportamiento inesperado en el componente que hace

la llamada. Por ejemplo, puede llamarse a una rutina de búsqueda binaria con un vector no ordenado para realizar la búsqueda. En este caso, la búsqueda podría fallar.

3. ***Errores temporales.*** Se producen en sistemas de tiempo real que utilizan una memoria compartida o una interfaz de paso de mensajes. El productor de los datos y el consumidor de dichos datos pueden operar a diferentes velocidades. A menos que se tenga un cuidado particular en el diseño de la interfaz, el consumidor puede acceder a información no actualizada debido a que el productor de la información no ha actualizado la información de la interfaz compartida.

5. Desarrollo dirigido de Pruebas

- ✓ Se comienza por identificar el incremento de funcionalidad requerido.
- ✓ Éste usualmente debe ser pequeño y aplicable en pocas líneas del código.
- ✓ Se escribe una prueba para esta funcionalidad y se implementa como una prueba automatizada.
- ✓ Esto significa que la prueba puede ejecutarse y reportarse, sin importar si aprueba o falla.
- ✓ Luego se corre la prueba, junto con todas las otras pruebas que se implementaron.
- ✓ Luego se implementa la funcionalidad y se opera nuevamente la prueba.
- ✓ Una vez puestas en funcionamiento con éxito todas las pruebas, se avanza a la implementación de la siguiente funcionalidad.

Universidad Nacional Arturo Jauretche
Ingeniería de Software I
Testing e Integración
Dr. Sergio D. Conde

6. Pruebas de Usuario

- ✓ Definir criterios de aceptación.
- ✓ Establecer Plan de Prueba.
- ✓ Ejecutar las pruebas de aceptación
- ✓ Ver resultados de las pruebas de aceptación.
- ✓ Realizar reportes de la prueba de aceptación

Descubriendo Testing

Dentro de los Testing tenemos diferentes tipos de fallas:

Fallas en el Desarrollo

- ✓ **Fallas de datos** Variables usadas antes de inicialización. Variables declaradas pero nunca usadas.
- ✓ Variables asignadas dos veces pero nunca usadas entre asignaciones. Posibles violaciones de límites de arreglo. Variables no declaradas.

- ✓ **Fallas de control** Código inalcanzable. Ramas incondicionales en ciclos.

- ✓ **Fallas entrada/salida:** Variables de salida dobles sin intervención de asignación. Comprende utilizar variables que no cumplen ninguna función.

- ✓ **Fallas de interfaz:**
- ✓ Incompatibilidad del tipo de parámetro.
- ✓ Incompatibilidad del número de parámetro.
- ✓ No se usan resultados de funciones.
- ✓ Funciones y procedimientos no llamados.

- ✓ **Fallas de gestión de almacenamiento:**
- ✓ Aritmética de almacenamiento.
- ✓ Fuga de memoria.

Integración

La integración continua es una práctica de desarrollo de software donde miembros de un equipo de desarrollo integran su trabajo de manera frecuente.

Los desarrolladores pueden una o varias veces por día lo que permite múltiples integraciones por día por parte del equipo de desarrollo.

¿ Como actúa?

Lo importante de cada una de las integraciones que realizan los desarrolladores es verificado por una compilación automática (automate build) incluido una batería de test.

El objetivo es la detección de errores en el sistema tan rápido como sea posible.

División general

Mantener el código en un solo lugar donde cualquier persona puede obtener los fuentes actuales y los anteriores

Automatizar el proceso de compilación / construcción de manera que permita que cualquier persona pueda hacer uso de un solo comando para compilar el código fuente del sistema.

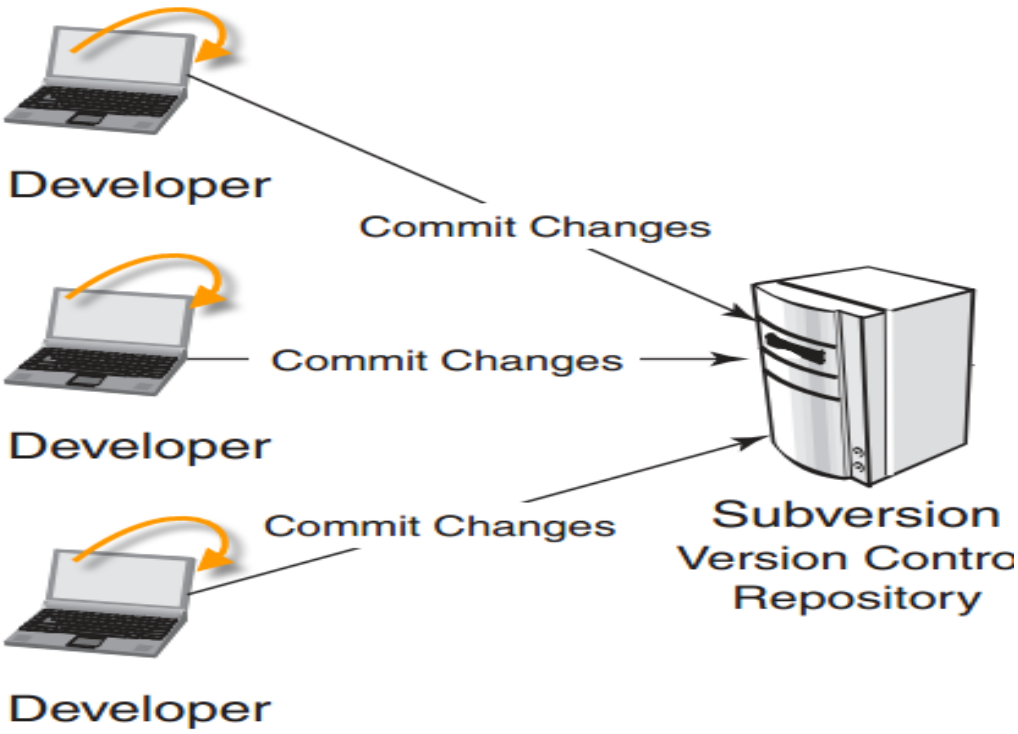
Automatizar las pruebas (test) de manera que pueda ejecutar un subconjunto de pruebas del sistema de un solo comando

Asegurarse de que cualquiera que quiera obtener una versión ejecutable obtenga el mejor que se pueda obtener en ese momento.

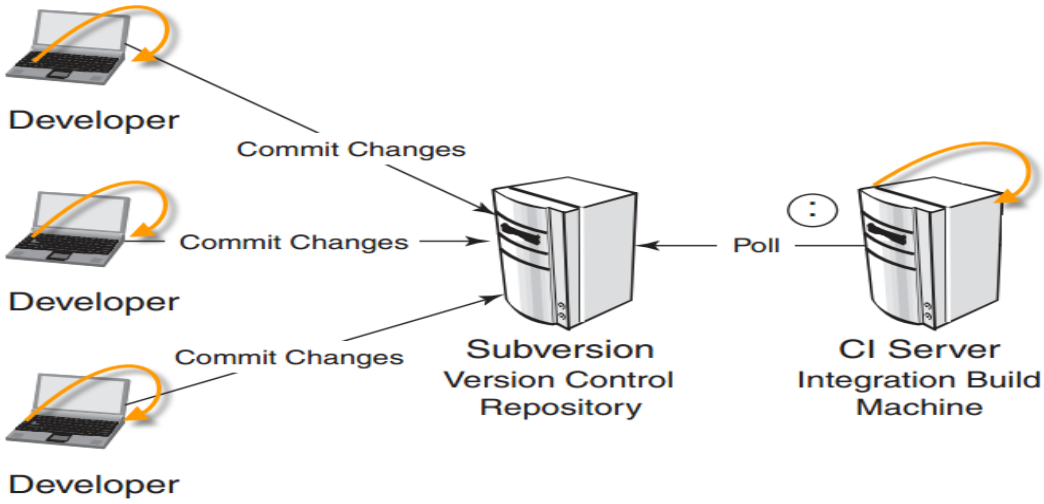
¿Cómo lo Hacemos?

Partimos de Desarrolladores:

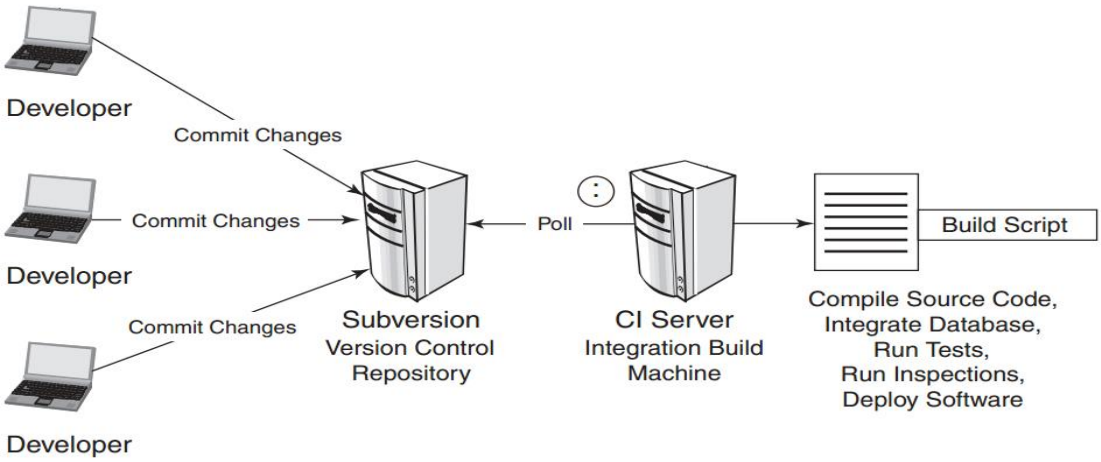
Agregamos una herramienta fundamental, el control de versiones


Developer
Commit Changes
Commit Changes
Developer
Commit Changes
Subversion
Version Contro
Repository
Developer

Ahora el servidor de integración continúa

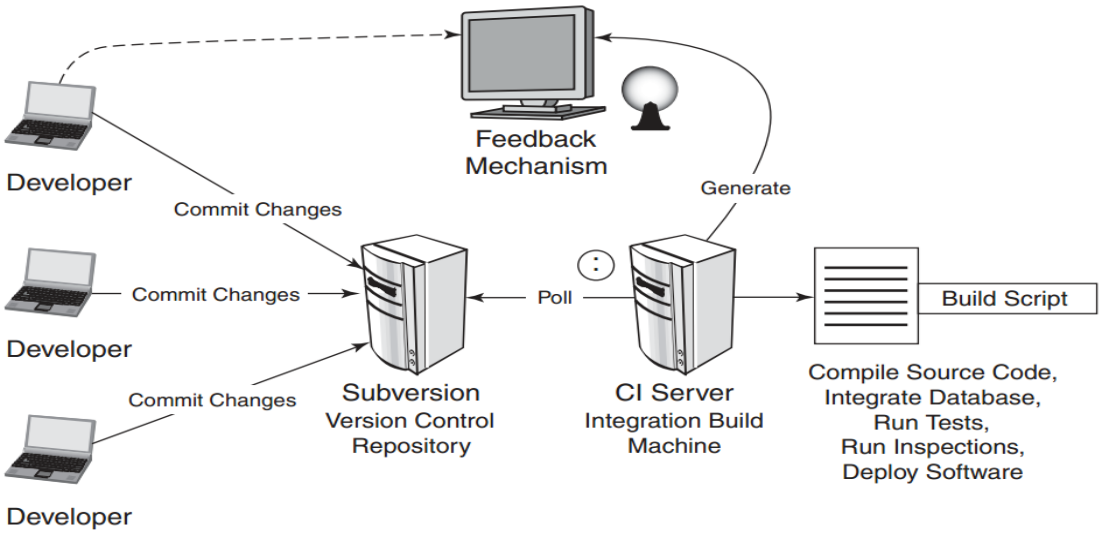


¿Qué tiene el servidor de integración continua?



Universidad Nacional Arturo Jauretche
Ingeniería de Software I
Testing e Integración
Dr. Sergio D. Conde

Una Prueba Visual



Esquema con Base de Datos