Programa de entrenamiento intensivo

PEI 2016

10 – Testing automático y deploy de la solución

baufest

Sobre los instructores

Leandro Goldin

Ingeniería en Sistemas de Información - UTN

.Net Technical Leader

2 años en Baufest

Principales proyectos en los que participé…

- Control y Gestión de Obras (AySA)
- Modernización Área de Investigación (Don Mario)

Sobre los instructores

Christian Smirnoff

Ingeniero en Informática – UADE - 2011

.Net Technical Leader

5 años en Baufest

Principales proyectos en los que participé…

- Comisiones (Falabella)
- Ajustes al Origen / Facturación por Módulos (La Nación)
- Evaluación de Productos de Desarrollo (Bunge)
- Transportation Management System (SC Johnson)
- Sincronización y Tareas de Galpón (Don Mario)
- Control y Gestión de Obras (AySA)
- Factory Desarrollo Club La Nación (La Nación)

Objetivos del Módulo

- Entender el concepto de testing del desarrollador
- Comprender qué es un test unitario automático
- Conocer buenas prácticas para diseñar y desarrollar aplicaciones testeables
- Obtener las herramientas necesarias para utilizar estas prácticas
- Aprender cómo realizar el deploy de una aplicación Web
- Conocer el concepto de integración continua

Agenda

- Testing del desarrollador
  - Introducción
  - Tests unitarios
  - Tests de integración
- Tests unitarios
  - Tips de arquitectura
  - Ejemplos de arquitectura
- Deploy de la solución
- Integración continua

Testing del desarrollador

Introducción

- ¿Qué es un test?
  - Es una prueba que compara el resultado esperado y el obtenido al ejecutar cierta funcionalidad de un sistema.

- ¿Qué es un test de desarrollador?
  - Código escrito por el desarrollador para testear que lo desarrollado genera los resultados esperados (caja blanca).
  - Es complementario a las pruebas funcionales, generalmente realizadas por un especialista en testing (caja negra).
  - Generalmente se ejecutan de forma automática mediante una herramienta.

Tests unitarios

- ¿Qué es un test unitario?
  - Es un test que se realiza de forma unitaria, es decir, abstrayendo el objeto a testear de sus dependencias con otros componentes.
  - Prueba el comportamiento del objeto a testear.

- Cómo escribir tests unitarios:
  - Setup de precondiciones
  - Ejecutar el código a testear
  - Realizar asserts sobre los resultados esperados

Tests unitarios

- Un buen test unitario:
  - Documenta el diseño de la aplicación
  - Tiene control total de todos los componentes en ejecución
  - Puede ejecutarse en cualquier orden si es parte de muchos otros tests
  - Retorna consistentemente el mismo resultado
  - Prueba un único concepto lógico en el sistema
  - Tiene un nombre claro y consistente
  - Es legible
  - Es mantenible

Tests de integración

- Los tests de integración:
  - Testean la “integración” entre componentes
  - Son complementarios a los tests unitarios
  - Usan dependencias tales como una base de datos
  - Pueden ser utilizados para probar stored procedures y llamadas a aplicaciones externas
  - Son menos performantes que los tests unitarios y a veces se ejecutan menos frecuentemente
  - Se enfocan en métodos con dependencias, no pruebas de la aplicación de punta a punta

Tests de integración

- Cómo escribir tests de integración:
  - Setup de precondiciones
  - Ejecutar el código a testear
  - Realizar asserts sobre los resultados esperados

Tests de integración

- Un buen test de integración:
  - **Utiliza dependencias de forma controlada**
  - Documenta el diseño de la aplicación
  - Puede ejecutarse en cualquier orden si es parte de muchos otros tests
  - Retorna consistentemente el mismo resultado
  - Prueba un único concepto lógico en el sistema
  - Tiene un nombre claro y consistente
  - Es legible
  - Es mantenible

Tests Unitarios: Tips y Ejemplos de Arquitectura

baufest

Tip 1: Programación orientada a interfaces

- Mantenibilidad
  - Permite cambiar la implementación interna de las clases concretas sin modificar el código de la aplicación.
- Extensibilidad
  - Permite la creación de diferentes clases concretas que implementen la interfaz sin modificar el código de la aplicación.
- Testeabilidad (cuando se usa en conjunto con Tip 2)
  - Permite el uso de clases Mock para testear componentes unitariamente.
  - El código de la aplicación no depende de clases concretas.

Tip 2: Inyección de dependencias por constructor

- Una inyección es el pasaje de una dependencia (un servicio) a un objeto dependiente (un cliente). El servicio es parte del estado del cliente. El cliente no crea ni busca el servicio.
- Requiere que el cliente provea un parámetro en un constructor para la dependencia.
  - public Constructor (IDependency dependency)
- El cliente ya no necesita ningún conocimiento sobre la implementación concreta que va a utilizar, favoreciendo la reusabilidad, testeabilidad y mantenibilidad.

Tip 2: Inyección de dependencias por constructor

- Se pueden inyectar objetos Mock para tests unitarios
- Adhiere al Dependency Inversion Principle (soliD)
- Hace obvias las violaciones al Single Responsibility Principal (Solid)
  - public Constructor(IClass1 c1, IClass2 c2, IClass3 c3, IClass4 c4, IClass5 c5, ……)
- Frameworks de inyección de dependencias (IOC)
  - Ninject, SimpleInjector, Castle, Autofac, Unity, Spring.NET…

Tip 3: Favorecer la composición por sobre la herencia

- Ideal para casos donde un tipo va a implementar solamente una parte del comportamiento expuesto por la superclase.
- Permite que las subclases implementen nueva funcionalidad sin afectar otras subclases.
- Permite cambios de comportamiento en tiempo de ejecución.

- *Elegir la composición por sobre la herencia ya que es más maleable y sencilla para la modificación de código, pero tampoco componer en todos los casos*

Tip 4: Generar Tests Unitarios

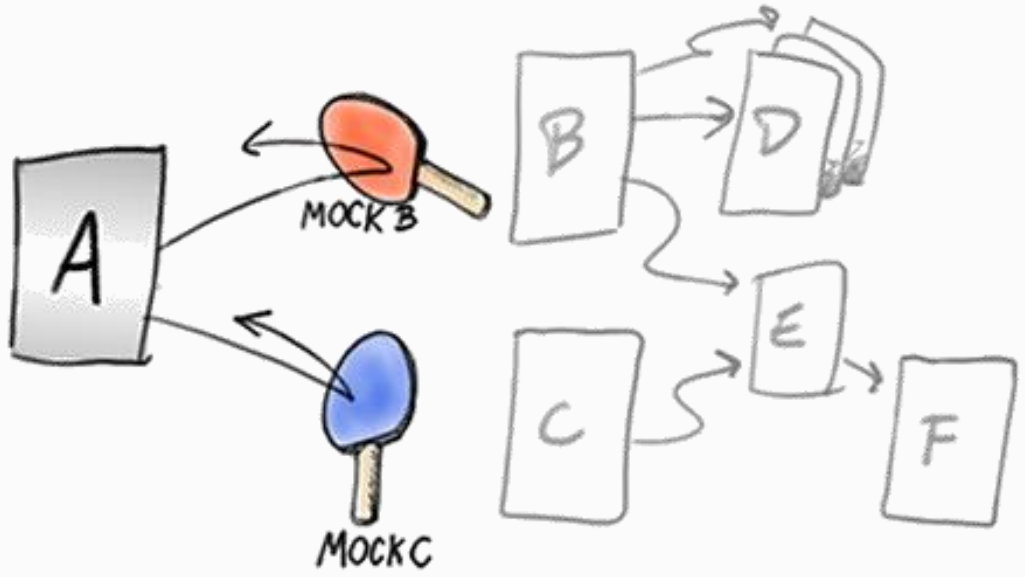
- Cómo escribir tests unitarios:
  - Setup de precondiciones
  - Ejecutar el código a testear
  - Realizar asserts sobre los resultados esperados

Tip 5: Mocking de dependencias

- No todo el código está contenido en sí mismo
- Un test unitario debe probar el código sin probar las dependencias
- Ver Tips 1 y 2
  - Programación orientada a interfaces
  - Inyección de dependencias por constructor

Tip 5: Mocking de dependencias

- Los objetos Mock son objetos simulados que imitan el comportamiento de objetos reales de forma controlada
- Permiten realizar verificación del comportamiento del objeto



Tip 5: Mocking de dependencias

- Objetos Dummy
  - Es un modelo simplificado de la implementación real
  - Son implementaciones concretas de la interfaz de la dependencia
  - Generalmente retornan resultados falsos y conocidos
  - No se utilizan en la aplicación productiva

Tip 5: Mocking de dependencias

- Cómo escribir tests unitarios usando Mocks:
  - Setup de precondiciones incluyendo el setup de los objetos mock
  - Inyectar mocks de dependencias
  - Ejecutar el código a ser testeado
  - Realizar asserts sobre los resultados esperados
  - Verificar que el mock fue llamado la cantidad de veces y con los parámetros esperados

Tip 6: Escribiendo código testeable

- No mezclar el grafo de instanciación de objetos con la lógica de la aplicación
- Pedir los objetos, no ir a buscarlos
- No escribir lógica en el constructor
- Tener cuidado con estado global y singletons
- Tener cuidado con métodos estáticos
- Elegir el polimorfismo por sobre los condicionales
- No mezclar objetos de servicio con objetos de valor
- No mezclar responsabilidades

Tip 7: Wrappers para encapsular dependencias estáticas

- Evitan acoplar el código directamente a librerías de terceros
- Se puede cambiar de librería de terceros sin cambiar el código de la aplicación
- Permiten usar mocks de dependencias **estáticas** de terceros
- Evitan el uso de clases específicas de librerías de terceros en tu código
- No **siempre** es necesario usar wrappers para dependencias de terceros

Deploy de la solución

Deploy de la solución

- El deploy de una aplicación consiste en las actividades que permiten que la misma esté lista para su uso.

- Se utiliza un servidor de aplicaciones, que permite crear y configurar aplicaciones web y provee el entorno para ejecutarlas.
  - Internet Information Services (IIS) - Microsoft

- Precondiciones:
  - Sitio Web creado en el servidor de aplicaciones
  - Configuración del sitio y su application pool
    - Credenciales, handlers, recycles, mappings, rutas, etc.

Deploy de la solución

- Pasos:
  - Compilar la solución con la configuración deseada (ej. Release - AnyCPU)
  - Generar un paquete con todos los archivos necesarios para instalar la aplicación (Publish)
  - Mover el paquete a la ruta asociada al sitio en el servidor de aplicaciones (IIS)
  - Acceder a la aplicación y comprobar su correcto funcionamiento

Integración continua

Integración continua

- Integración continua es una práctica, adoptada por eXtreme Programming (XP), que consiste en la unificación de las versiones de la aplicación de cada desarrollador a una línea base compartida, varias veces al día.
- Se utiliza en combinación con pruebas unitarias y de integración automatizadas, que se ejecuten periódicamente o ante cada actualización al repositorio de código, reportando los resultados al equipo de desarrollo.

Integración continua

Algunos de los beneficios que la práctica promueve son:

- Permite que los errores sean detectados de forma temprana en el contexto de desarrollo y localizarlos fácilmente
- Reduce tiempo de desarrollo y mantenimiento durante la vida de un proyecto
- Mejora la calidad del producto final, entregándole software con menos errores al cliente
- Mantiene disponible siempre una versión actual de la aplicación para testing, demo o despliegue
- Mejora la organización del equipo de desarrollo
- Permite generar métricas de calidad de código, cobertura de pruebas automatizadas, etc.
- Puede asociarse a prácticas como el despliegue continuo, para la instalación de la aplicación en otros ambientes

¿Preguntas?

baufest

¡Muchas Gracias por Participar!

baufest