UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

REESTRUCTURACIÓN Y REFACTORIZACIÓN DE UNIT TESTS CON TESTSURGEON

MEMORIA PARA OPTAR AL TÍTULO DE TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

PABLO IGNACIO ESTEFO CARRASCO

PROFESOR GUÍA:
ALEXANDRE BERGEL

MIEMBROS DE LA COMISIÓN:
ROMAIN ROBBES
JOSÉ PINO URTUBIA

SANTIAGO DE CHILE
JULIO 2013

Shegó el resumeeeen

Por mi y por todos mis compañeros.

Agradecimientos

Vale cabros :D

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Capítulo 2

Especificación del Problema

2.1. Motivación

2.1.1. Contexto: Test como "conductores" del diseño

Durante el desarrollo de un software uno de las actividades principales es el testeo de los requerimientos. Existen variadas técnicas, metodologías y artefactos relacionados a esta actividad. En particular, la creación de pruebas automatizadas es una práctica cotidiana en cualquier proyecto de software e inclusive es actualmente considerado parte de los artefactos generados durante éste.

En particular las metodologías ágiles dan una importancia fundamental a los tests de tal manera que está prohibido agregar una nueva funcionalidad sin, previamente, haber escrito un test que lo valide[**??**]. Esta queda documentado en el libro de Robert Cecil Martin (más conocido como"Uncle Bob")[**??**] uno de los escritores del manifiesto ágil[**??**] quien declara:

> *"The iteration between writing test cases and code is very rapid [...]. As a result, a very complete body of test cases grows along with the code."*

Uno de los ejemplos más conocidos de implementación de estos principios es la metodología de Desarrollo Dirigido por Pruebas o *Test-Driven Development*[**??**] el cual propone que la implementación de una funcionalidad del software debiera seguir los siguientes pasos:

1. Escribir un test que valide el funcionamiento esperado de la nueva característica
2. Escribir código base mínimo que haga pasar dicho test
3. Refactorizar el código

En la práctica, aplicando TDD se obtiene una gran cantidad de pruebas unitarias (unit tests) que representan los casos de prueba de la aplicación (Test Cases). Cada Unit Test

contiene varios métodos de tests (test methods) que cubren los distintos aspectos a verificar en la funcionalidad que se está testeando.

2.1.2. Problema: Muchos tests, mal diseñados y lentos.

La comunidad de ingeniería de software ha producido herramientas efectivas y buenas prácticas para lograr refactorizaciones en el código base que otorguen un buen diseño de código. Sin embargo, en cuanto al código de los tests unitarios no es así. De hecho, estos son raramente modificados y carecen del cuidado que se le da al código base y no se piensa en su modularidad ni extensibilidad.

Por lo cual no es difícil encontrar deficiencias como **solapamientos** entre test methods o más general, entre test cases. Estos solapamientos pueden ser de carácter estático: duplicación de código, o bien dinámicos, es decir que dos tests methods tienen ejecuciones similares y por consiguiente testean lo mismo.

Estas deficiencias en el diseño y calidad del código de las pruebas unitarias tiene consecuencias importantes en la calidad del código testeado y en el mismo proceso de desarrollo:

- **Performance** Debido a los solapamientos previamente mencionados, muchas veces existe **ejecución redundante** que va en contra de las características deseables de una suite de tests[**??**]. Muchas veces los tests dejan de ejecutarse con la frecuencia deseada.
- **Debugging** Otra deficiencia conocida es cuando al correr los tests en presencia de un defecto, éste se queda en evidencia por muchos tests methods, lo cual dificulta la identificación de la causa del bug y su corrección. Coloquialmente se hace más difícil responder la pregunta: *¿Cuál test miro primero?*

De esta manera la confiabilidad en el producto y su calidad se ven impactadas negativamente.

Un caso muy claro del impacto del mal diseño y poco cuidado en los tests sucede en la práctica de **Integración Contínua** (Continuous Integration)[**??**]. En esta práctica cada *feature* se implementa tan pronto como es posible, se testea y se pasa a producción de inmediato. De esta manera el equipo de desarrollo realiza varias integraciones por día y el cliente obtiene rápidamente las nuevas funcionalidades a medida que las solicita.

A modo de ejemplo, la empresa MediaGeniX[1] realiza la integración continua desde hace un tiempo. Ahí, 30 desarrolladores trabajan sobre el mismo producto y cada uno de ellos construye varias versiones al día. Cada versión que se va a pasar a producción debe pasar su suite de pruebas que comprende alrededor de 30.000 tests. Ellos realizan al menos 3 integraciones diarias en promedio por lo cual recurren a técnicas de paralelización de ejecución de tests para lograrlo. Esto introduce un alto costo económico asociado al los recursos de hardware(servidores, clusters) y además un alto costo en tiempo.

[1] MediaGeniX, `http://www.mediagenix.tv`

Esto evidencia la relevancia de mejorar la performance mediante la refactorización y restructuración de los unit tests.

Capítulo 3

Trabajo Relacionado

Capítulo 4

Descripción del Problema

Capítulo 5

Validación de la Solución

Capítulo 6

Conclusión