Universidad de Buenos Aires

FACULTAD DE INGENIERÍA

Departamento de Computación

75.10 – Técnicas de diseño

TRABAJO PRÁCTICO 2.3

AGREGADO DE FUNCIONALIDAD A UN CÓDIGO EXISTENTE

Curso: *2013 – 2do Cuatrimestre*

GRUPO Nº 13		
APELLIDO, Nombre	**Número de Padrón**	**e-mail**
Barrea, Ignacio	*86225*	*ignacio@tictaps.com*
Chavar, Hugo	*90541*	*hechavar@gmail.com*
Schmoll, Edward Erik	*90135*	*erikschmoll@gmail.com*

Informe: TP 2.3

Críticas al grupo 11

- No estaba implementada la funcionalidad para obtener xml.
- No funcionaba correctamente ya que los test se mostraban Ok en el reporte cuando no pasaron.
- En los setters se hacían concatenaciones:

```
@Override
public void setTiempoEjecucion(double tiempo) {
	tiempoEjecucion = tiempo;
	errorMsg = errorMsg + " (" + tiempoEjecucion + " ms)";
}
```

Debido a esto nos fue muy costoso hacer cambios aunque sean pequeños en esa clase. El cambio que se necesito hacer es que se cree a partir de un xml, y esto nos llevo a cambiar casi todos los métodos de la clase.

- Hay métodos que retornan null:

```
public String getCollectionResultCadenaDeNombres() {
	return null;
}
```

En particular este nos hizo perder mucho tiempo ya que se informaba como mensaje de error un nulo.

```
public static void validateTrue(boolean condition) {
	if (!condition) {
		informFail(null);
	}
}
```

- Hay tests que es imposible que fallen ya que son pasados por valor a un método los valores que luego son comparados, que además no hace modificaciones sobre esos datos:

```
@Test
public void testValidateEqualsIntInt() {
	int condicion1 = 1;
	int condicion2 = 1;
	Validation.validateEquals(condicion1, condicion2);
	assertEquals(condicion1, condicion2);
}
```

- No habían ejemplos de tests que fallan (O sea, ninguno de los que estaban terminaban en [Fail]).

- No habían tests que mostraran cómo funcionan las excepciones(O sea, ninguno de los que estaban terminaban en [Error]).

Esto nos limitó en los tests que pudimos escribir ya que esta funcionalidad no la pudimos probar.

- Las clases Fixture, FixtureData, RunAll, RunRegularExpression, RunTags, RunTagsWithRegExp y RunTemplate no estaban en el diagrama de clases.

- No estaban separadas en paquetes de java las diferentes funcionalidades (grabar archivos y UnitTest compartían el mismo package).

- No estaban separados en packages los tests hechos con JUnit de los que estaban hechos con su propio framework.

- El usuario del framework tiene que ingresar muchos detalles como crear Runners, que se podrían haber encapsulado en un solo método:

```
RunTemplate runMethod = new RunTags(TagType.SLOW);
testSuite.setRunMethod(runMethod);
```

Se podría haber implementado un metodo que el usuario invoque de la siguiente manera:

```
testSuite.addTagToRun(TagType.SLOW);
```

- Tienen una clase Singleton para registrar nombres de tests que lo juzgamos innecesario y solo agrega complejidad, hacía que no se puedan crear suites diferentes con mismos nombres de tests sin invocar a un clear en ese singleton. Al menos no se entiende el sentido de esto, y por falta de tiempo (Si lo sacábamos muchos tests empezaban a fallar) decidimos dejarlo como estaba invocando a este singleton para que limpie la lista, aunque nuestro código quede poco claro.

- Otro metodo que devuelve null en ciertas situaciones, tuvimos que aceptar esto por falta de tiempo y preguntar si no es null antes de ejecutar la acción:

```
@Override
public TestResult run(GenericTest test) {
      TestResult result = null;
      if (test.getName().matches(regExp) &&
containsTag(test.getTags())) {
            result = test.run();
      }
      return result;
}
```

- No pudimos resolver por falta de tiempo los System.out.println() que habían por todos lados, y decidimos usar lo mismo.

El método run tiene todas estas líneas que no aportan a la claridad de lectura del mismo:

```
String contenedoraYCollectionActual;
if (nombreContenedora == null) {
      contenedoraYCollectionActual = getName();
} else {
      contenedoraYCollectionActual = nombreContenedora + "." +
getName();
}
System.out.println(" ");
System.out.println(contenedoraYCollectionActual);
```

- Se mezcla mucho el español y el inglés en los nombres de métodos y atributos.

- No almacenaba información sobre los métodos sobre los que se hizo un skip, y nos complicó a la hora de levantar los tests desde los stores ya que no había forma de diferenciarlos. Y no era sencillo hacer el cambio ya que ellos los salteaban en la ejecución.

Resolución de nuevas funcionalidades

Para la creación a traves de xml se descarta el uso de patrones creacionales como Builder, Prototype y Abstract Factory debido a que se debe construir el objeto a través del xml, requiriéndose recursividad, para resolverlo se agrega un método estático que recibe el xml y devuelve el TestCollection (TestSuite).

Se esperaba hacer un patrón Strategy para resolver los stores pero fue muy complicado desacoplar las cosas que ya estaban funcionando con tests hechos, por eso agregamos a TestCollection variables indicadoras para los casos en que se desee usar un store.