



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2013 - 2^{do} Cuatrimestre

TÉCNICAS DE DISEÑO (75.10)

TRABAJO PRÁCTICO N°2.2
FECHA: 31/10/2013

INTEGRANTES:	PADRON
Franetovich, Damian <damian168@gmail.com>	88924
Rial, Sebastián <riseba@gmail.com>	90309
Pivetta, Agustín <agus pivetta@gmail.com>	90789

Índice

1. Enunciado	2
2. Diagramas	4
3. Justificaciones	6

1. Enunciado

Objetivo

- Taggear tests (un string, ejemplo: FAST, SLOW, DB, INTERNET, etc)
- Poder agregar más de un tag a un test.
- Permitir desactivar un test con SKIP.
- Poder ejecutar test pertenecientes a un tag particular.
- Poder ejecutar test pertenecientes a varios tags.
- Poder ejecutar test pertenecientes a varios tags y/o que su nombre de test case coincida con una expression regular y/o que su nombre de test suite coincida con una expresión regular.
- Tomar el tiempo a todos los test para poder agregarlos en reportes.
- Reporte textual por línea de comando. (Utilizar formato de la entrega anterior).
- Que el reporte textual por línea de comando anterior sea progresivo en real time (a medida que vaya teniendo resultados los muestra).
- Implementar un reporte xml siguiendo el siguiente XSD: <https://svn.jenkinsci.org/trunk/hudson/dtkit/dtkitformat/dtkitjunitmodel/src/main/resources/com/thalesgroup>

Formato de salida del Reporte XML

Ver:

<https://svn.jenkinsci.org/trunk/hudson/dtkit/dtkitformat/dtkitjunitmodel/src/main/resources/>

Restricciones

- Trabajo Práctico grupal implementado en java o C#
- Se deben utilizar las mismas herramientas que en el TP0 (git + maven + junit4 / git + VS 2012 + MS Test o NUnit).
- Todas las clases del sistema deben estar justificadas.
- Se debe modelar utilizando un modelo de dominio, y no usando herramientas tecnológicas como reflection, annotations, etc.

- Todas las clases deben llevar un comentario con las responsabilidades de la misma.
 - El uso de herencia debe estar justificado. Se debe explicar claramente el porqué de su conveniencia por sobre otras opciones.
 - Se debe tener una cobertura completa del código por tests
 - Se pide además de tener los test unitarios junit/nunit.
- * No se aceptaran TP?s que violen alguna de las restricciones.

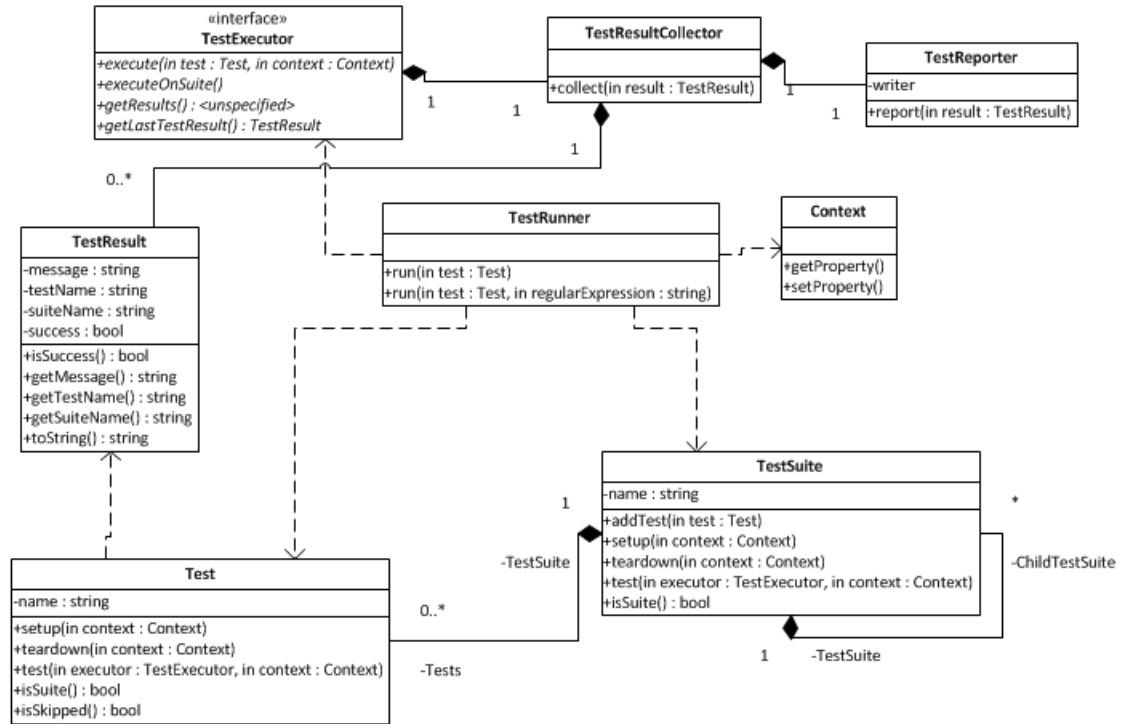
Criterios de Corrección

- Cumplimiento de las restricciones
- Documentación entregada
- Diseño del modelo
- Diseño del código
- Test Unitarios

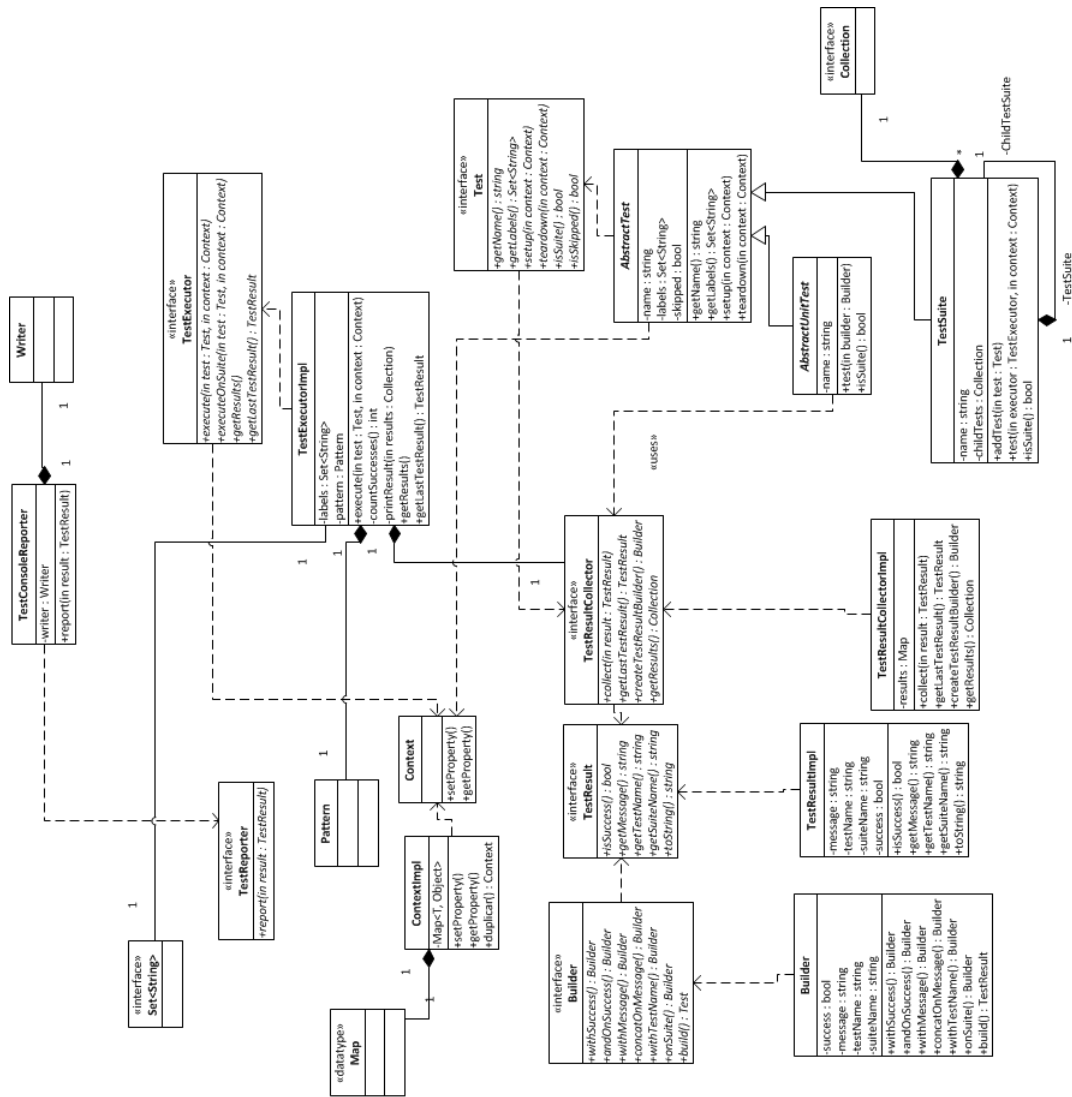
Se tendrán en cuenta también la completitud del tp, la correctitud, distribución de responsabilidades, aplicación y uso de criterios y principios de buen diseño, buen uso del repositorio y uso de buenas prácticas en gral.

2. Diagramas

75.10
Tp N°2
Diagramas de Clase de Dominio



Diagramas de Clase de Diseño



3. Justificaciones

- **Herencia:** Abstract Test, AbstractUnitTest, TestSuit

Motivación:

Reducir la duplicidad de código. Los AbstractUnitTest y los TestSuit tienen características comunes a ambos, las cuales agrupamos en la clase AbstractTest. En principio solo encontramos la necesidad de un nombre como característica común, pero podría haber más, entonces esto ayuda también a encapsular el cambio.

- **Patrones utilizados:**

Builder:

Utilizado en: TestResult.java, clase ?Builder?

Motivación: Se requería construir un TestResult (resultado de un test), como resultado de distintos eventos que modificaban el estado del mismo. El Builder permite hacer esto de una forma clara.

Composite:

Utilizado en: Test, TestSuite

Motivación: La forma en que se organizan los test con la estructura de un árbol presentan una oportunidad de utilizar este patrón beneficiarnos de sus ventajas y sin tener que lidiar con sus desventajas (la interfaz de los componentes es válida tanto para elementos hoja como para elementos no hoja).

Visitor:

Utilizado en: TestResultCollector

Motivación: Simplifica la recolección de resultados en la jerarquía de tests.

Command:

Utilizado en: TestRunner

Motivación: Simplificación de interfaz de runner.

- **Clases:**

Las justificaciones pueden encontrarse en el código como comentario en cada clase, aquí se expone un resumen de las mismas.

Interfaces:

Test:

Interfaz genérica para un test.

TestResult:

Segregación de interfaz.

TestResultCollector:

Separar implementacion del colector de resultados.

TestRunner:

Segregacion de interfaz.

TestReporter:

Interfaz de Test Reporter

Assert1:

De esta forma los asserts que tengan el tipo de firma de Assert1 (recibiendo solo 1 parametro) solo deben redefinir el metodo apply para poder ser utilizados.

Assert2:

Igual Assert1.

Context:

Persistencia a travez de la corrida de los tests, de los elementos creados en los Setup.

Clases Concretas:

AssertXXXXX:

Implementan los distintos tipos de verificación.

Asserts:

Permite utilizar los asserts sin tener que crear instancias de los mismos.

AbstractUnitTest:

Implementacion para los tests unitarios.

TestResultCollectorImpl:

Implementacion del colector de resultados. Para poder obtener todos los resultados en el mismo lugar, en el caso de estar corriendo una estructura tipo arbol de tests.

TestResultImpl:

Implementación de Resultado de tests, tiene cierta información que se cree de utilidad para quien ha corrido los tests.

TestRunnerImpl:

Implementación del TestRunner, permite correr un test o suite de tests y obtener un resumen sobre los resultados.

TestSuite:

Contiene una colección de tests, permite organizar y correr tales tests, y tambien permite correr otros TestSuites, lo que otorga la posibilidad de correr los tests con una estructura del tipo arbol.

ContextImpl:

Implementación de Persistencia de contexto.

TestConsoleReporter:

Implementación del reporte progresivo de resultados de tests XmlMapper:

Mapeador de las clases a XML.