



Grupo de Investigación en  
Reutilización y Orientación a  
Objeto

# Refactorizaciones en la Migración del Software

Autores:

Raúl Marticorena  
Yania Crespo  
Carlos López

[rmartico@ubu.es](mailto:rmartico@ubu.es)  
[yania@infor.uva.es](mailto:yania@infor.uva.es)  
[clopezno@ubu.es](mailto:clopezno@ubu.es)

# Índice

- Problema
- Estado del Arte
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro



1. Problema
2. Trabajos Relacionados
3. Planteamiento del Problema
  - Evolución del Lenguaje
  - Evolución de Frameworks
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro



# Problema

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

3/17



## ■ Refactoring

- Tarea habitual en el desarrollo del software
- Objetivos
  - **Mejorar la comprensión**
  - **Facilitar el mantenimiento**
  - **Preservar el comportamiento**

## ■ Problema detectado:

- Migración entre versiones de:
  - Bibliotecas
  - Frameworks
  - Herramientas
- Cambios sugeridos por:
  - Mejoras del lenguaje (especificación) → mejoras en la nueva versión
  - Con preservación del comportamiento entre versiones

## ■ Caso de estudio particular: Java y JUnit

- Papel de las refactorizaciones en estos contextos



# Trabajos Relacionados

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

4/17



## ■ Migraciones:

- Anotaciones → Java & JUnit [Tansey & Tilevich]
- Genericidad → Java [Tip] [Fuhrer][Donovan]
- Soluciones ad-hoc

## ■ *Moose*

- Entorno de soporte a reingeniería
- Sin reutilización en la parte del motor

## ■ *RefaX*

- Basado en XML como soporte básico
- Implementación particular para las acciones

# Planteamiento del Problema

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

5/17



- Evolución del Lenguaje: especificaciones cambiantes
  - Ejemplo: Java – especificación 3.0
    - Genericidad, anotaciones, enumeraciones, importaciones estáticas, etc.
  - Ejemplo: .NET – Framework 2.0
    - Genericidad, etc.
- Implican posibilidades de reescritura de código
  - Mejorando la calidad del código (facilidad de mantenimiento)
  - Misma funcionalidad
- Ej: en JUnit 4.x se utilizan: anotaciones e importaciones estáticas

# Planteamiento del Problema

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

6/17



- Evolución del Framework
  - La evolución del lenguaje puede conducir a evolucionar el framework
  - Nuevas formas de resolver antiguos problemas
- Frameworks de caja blanca evolucionando a frameworks de caja negra o gris
  - Del uso intensivo de herencia a otros mecanismos declarativos
- Caso de estudio:
  - JUnit 3 con herencia y convención de nombres
  - JUnit 4 con anotaciones e importaciones estáticas

# Planteamiento del Problema

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

7/17

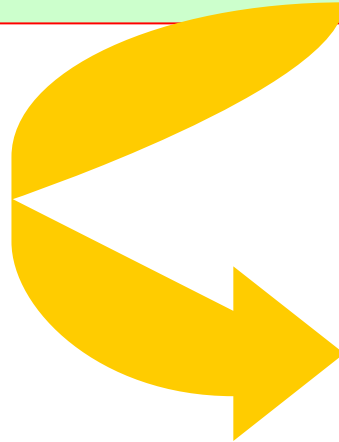


## ■ Ejemplo:

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class ListTest extends TestCase {
    public void setUp() {...}
    public void testCapacity() {...}
    public void testElementAt() {
        Integer i= (Integer)fFull.elementAt(0);
        assertTrue(i.intValue() == 1);
        try {
            fFull.elementAt(fFull.size());
        } catch (ArrayIndexOutOfBoundsException e) {
            return;
        }
        fail("Should raise an ArrayIndexOutOfBoundsException");
    }
    public void tearDown() {...}
}
```

JUnit 3.x

JUnit 4.x



```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import org.junit.After;
public class ListTest {
    @Before public void setUp() {...}
    @Test public void testCapacity() {...}
    @Test(expected = ArrayIndexOutOfBoundsException.class)
    public void testElementAt() {
        Integer i = (Integer) fFull.elementAt(0);
        assertTrue(i.intValue() == 1);
        fFull.elementAt(fFull.size());
        return;
    }
    @After public void tearDown() {...}
}
```



# Soporte al Código Fuente

## Marco del Trabajo

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

8/17



- Usando una plantilla de definición de refactorizaciones
  - Elementos identificados:
    - pre / postcondiciones (definidas con predicados y funciones)
    - acciones de transformación
- Aplicados sobre:
  - representación del código en un lenguaje minimal: MOON
  - o como extensión de MOON representando un lenguaje concreto
- Utilización de frameworks:
  - para la instanciación concreta a cada lenguaje
  - definición de un motor de ejecución de los elementos de las refactorizaciones



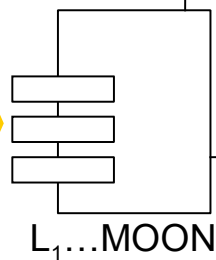
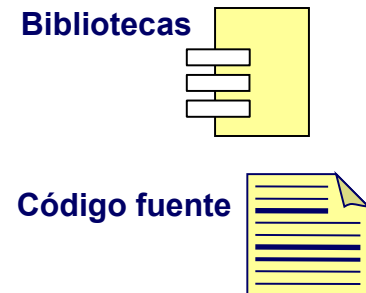


# Soporte al Código Fuente

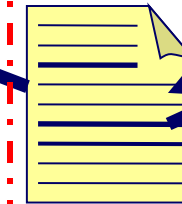
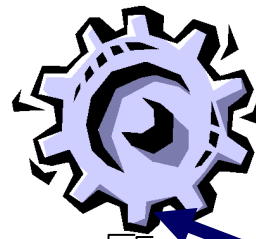
## Marco del Trabajo

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

9/17

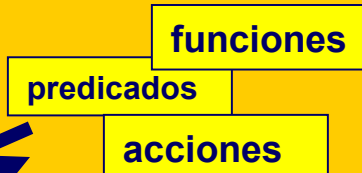


### Motor de Refactorizaciones



**Definición de Refactorización**

### Repositorio





# Soporte al Código Fuente

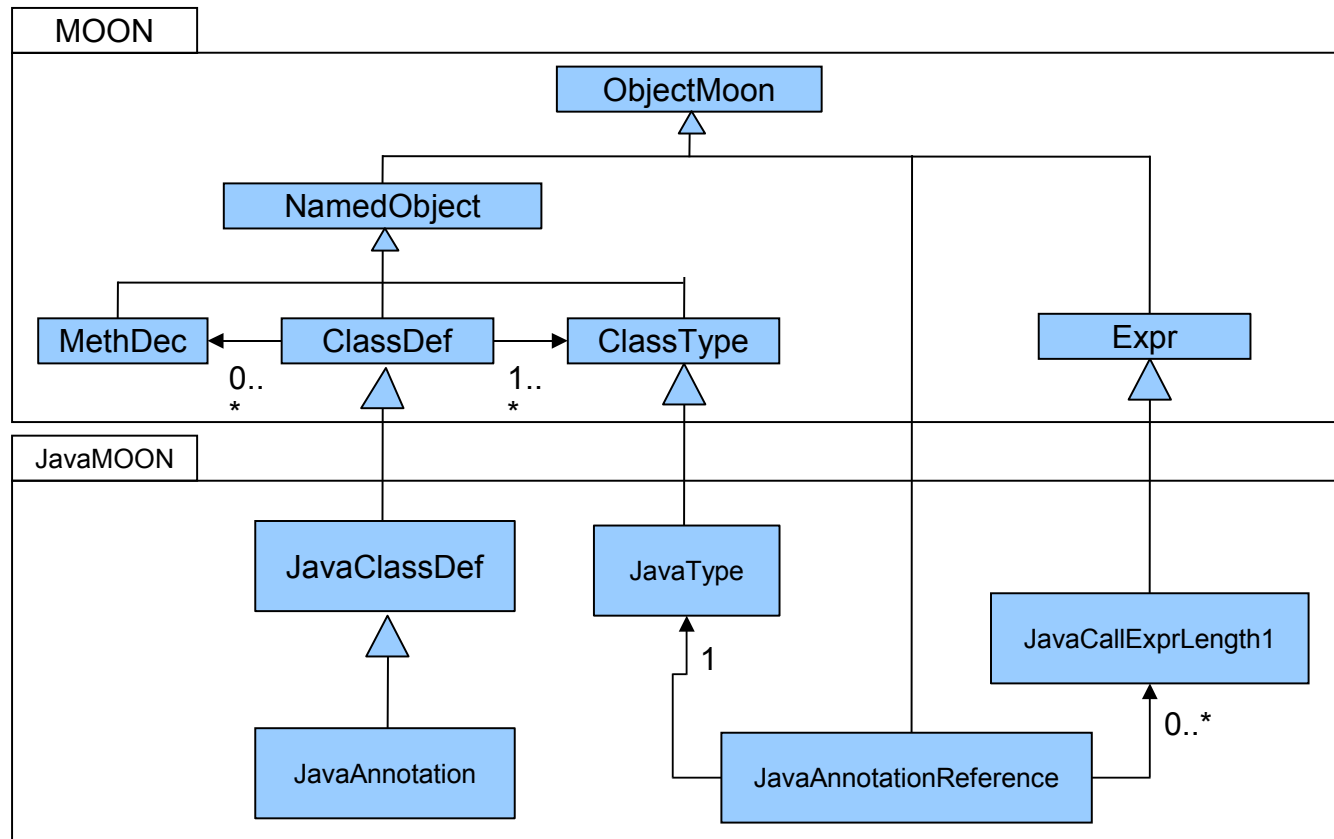
## Marco del Trabajo

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

10/17



- Modificaciones al modelo
  - Reutilizando MOON como base
  - Integrando los nuevos elementos en la jerarquía



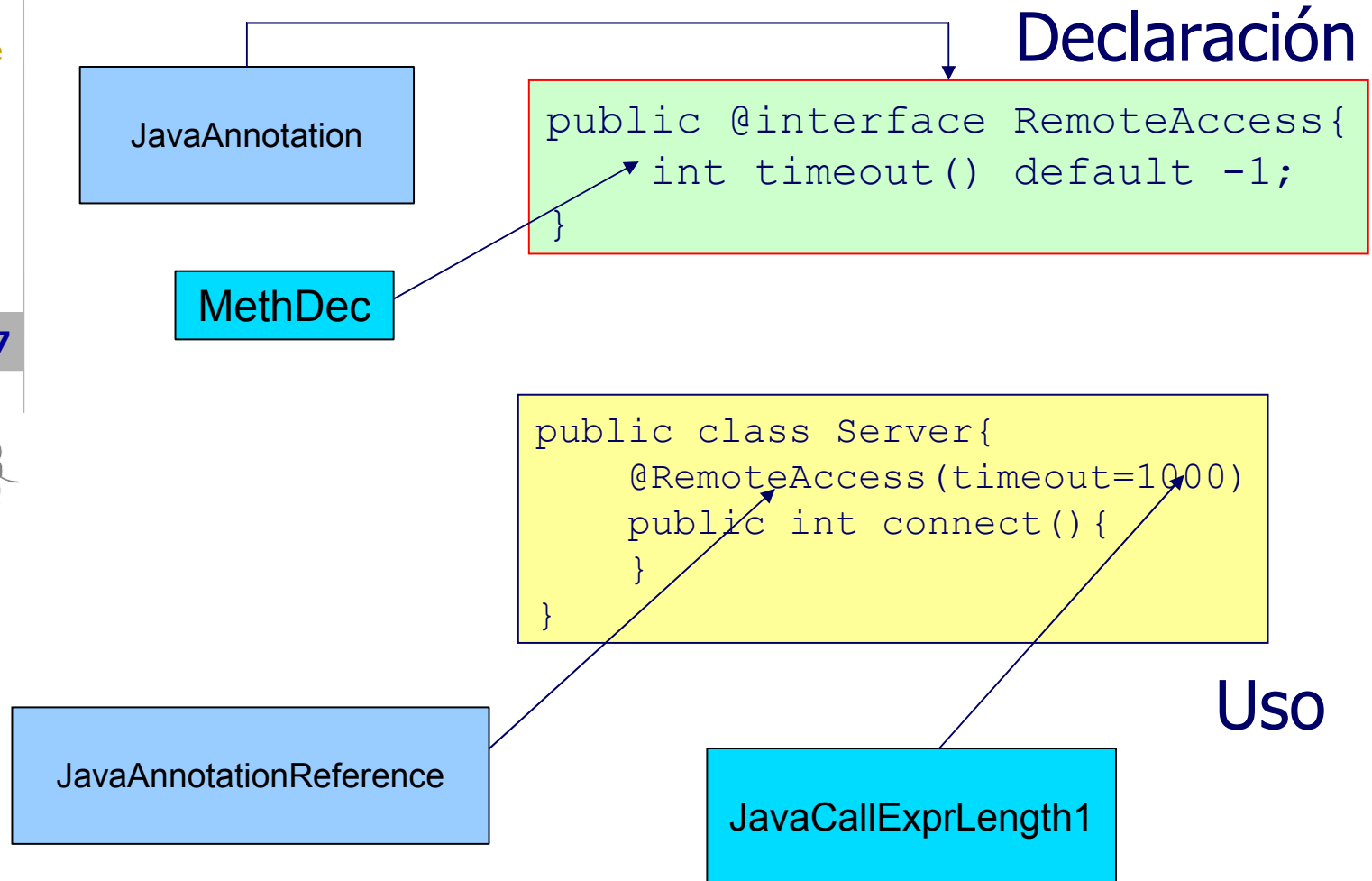
# Soporte al Código Fuente

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

11/17



## ■ Ejemplo:



# Soporte a Refactorizaciones para la Migración

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

12/17



## ■ Refactorizaciones planteadas:

### ■ A) Migrate Class From JUnit3 to JUnit4

- Migra una clase (conjunto de tests) de una versión 3 (herencia, etc.) a una versión 4 (anotaciones)
- Entrada: clase
- Precondiciones:
  - Hereda de `TestCase`
- Acciones
  - 1. Eliminar importaciones de la versión 3
  - 2. Añadir importaciones de la versión 4
  - 3. Eliminar cláusula de herencia
  - 4. Añadir anotaciones según convención de nombres
- Postcondiciones
  - No hereda de `TestCase`

# Soporte a Refactorizaciones para la Migración

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

13/17



## ■ Refactorizaciones planteadas:

### ■ B) Migrate Exception Method

- Migra un metodo (test) con captura de excepciones a una versión 4 con valor `expected` para la anotación
- Entrada: método
- Precondiciones:
  - El método es un test
- Acciones
  - 1. Añadir valor `expected` y cláusulas `throws`
  - 2. Eliminar `try-catch-finally`
  - 3. Eliminar `fail`
- Postcondiciones
  - El método es un test

# Soporte a Refactorizaciones para la Migración

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

14/17



## ■ Reutilización del esquema planteado:

- 1) Implementando nuevas acciones, funciones y predicados en los repositorios
- 2) Ensamblando los elementos:
  - Solución 1: programando la refactorización (framework de caja blanca)

```
public MigrateClassFromJUnit3ToJUnit4(ClassDef classDef, Model model) {  
    super(NAME, model);  
    // Preconditions  
    this.addPrecondition(new IsSubtype(classDef.getClassType(),  
                                       classType));  
  
    // Actions  
    this.addAction(new RemoveJUnit3Imports(classDef));  
    this.addAction(new AddJUnit4Imports(classDef));  
    this.addAction(new RemoveInheritanceClause(classDef, classType));  
    this.addAction(new AddJUnit4Annotation(classDef));  
    // Postconditions  
    this.addPostcondition(new IsNotSubtype(classDef.getClassType(),  
                                           classType));  
}
```

# Soporte a Refactorizaciones para la Migración

- Problema
- Trabajos Relacionados
- Planteamiento del Problema
- Soporte al Código Fuente
- Soporte a Refactorizaciones para la Migración
- Conclusiones y Líneas de Trabajo Futuro

15/17



- Solución 2: generando un fichero con la definición (XML) que se pueda ejecutar posteriormente (framework de caja gris)

```
<inputs>
  <input type="moon.core.Model" name="Modelo" root="false" />
  <input type="moon.core.classdef.MethDec" name="Method" root="true" />
</inputs>
<mechanism>
  <preconditions>
    <precondition name="IsJUnit4TestMethod">
      <param name="Method" />
    </precondition>
  </preconditions>
  <actions>
    <action name="AddJUnit4AnnotationTestExceptionValue">
      <param name="Method" />
    </action>
    <action name="RemoveTryCatchFinally">
      <param name="Method" />
    </action>
    <action name="RemoveJUnit3FailInstructions">
      <param name="Method" />
    </action>
  </actions>
  <postconditions>
    <postcondition name="IsJUnit4TestMethod">
      <param name="Method" />
    </postcondition>
  </postconditions>
</mechanism>
```

# Conclusiones y Líneas de Trabajo Futuro

- Problema
- Estado del Arte
- Marco del Trabajo
- Identificación de Elementos
- Soluciones Estáticas
- Evolución a Soluciones Iterativas e Incrementales
- Conclusiones y Líneas de Trabajo Futuro

16/17



## ■ Conclusiones

- Uso válido de refactorizaciones en la migración de software
  - Condición de preservación de comportamiento
- Validez de la solución basada en frameworks para la construcción y ejecución de refactorizaciones
  - Aportando distintas soluciones
  - Con cierto grado de reutilización de los elementos
- Aplicación de las refactorizaciones detectadas sobre los clientes mediante el motor

## ■ Líneas de trabajo futuro

- Mejora del soporte de código
- Aplicación en problemas similares
  - Ej: EJB nueva especificación basada en anotaciones
  - Ej: .NET ofrece el soporte de atributos equivalente a las anotaciones en Java



# Gracias por su atención



Autores: Raúl Marticorena  
Yania Crespo  
Carlos López

[rmartico@ubu.es](mailto:rmartico@ubu.es)  
[yania@infor.uva.es](mailto:yania@infor.uva.es)  
[clopezno@ubu.es](mailto:clopezno@ubu.es)



Grupo de Investigación en  
Reutilización y Orientación a  
Objeto