

CARACTERIZACIÓN DE REFACTORIZACIONES PARA LA IMPLEMENTACIÓN EN HERRAMIENTAS*

Carlos López^{1*}, Raúl Marticorena¹ y Yania Crespo²

1: Grupo GIRO. Área de Lenguajes y Sistemas Informáticos

Escuela Politécnica Superior

Universidad Burgos

09006 Burgos, España

e-mail: { clopezno, rmartico }@ubu.es, web: <http://www.giro.infor.uva.es/>

2: Grupo GIRO. Departamento de Informática

Escuela Técnica Superior de Ingeniería Informática

Universidad de Valladolid

Campus Miguel Delibes 47001 Valladolid, España

e-mail: yania@infor.uva.es, web: <http://www.giro.infor.uva.es/>

Palabras clave: refactoring, caracterización, cuantificación, características de desarrollo.

Resumen. *Aunque existen varias clasificaciones de refactorizaciones, ninguna está orientada a guiar su implementación. Este trabajo establece unos criterios de caracterización considerando el proceso completo de refactorización. A través de éstos, se asiste a la selección de un conjunto de refactorizaciones relacionadas, así como un orden de implementación de las mismas. Se proporciona un modelo de caracterización abierto para cualificar y cuantificar la complejidad de las refactorizaciones incluidas en distintos catálogos. Utilizando el modelo se presenta un caso de estudio, para guiar la decisión de implementación relacionada con el orden de implementación y selección de refactorizaciones asociadas con un conocimiento declarativo.*

1. INTRODUCCIÓN

Dentro de las actividades del proceso de refactorización se incluyen: la identificación de zonas del sistema a refactorizar, determinación de un conjunto de refactorizaciones a aplicar, aplicación del conjunto elegido, computar los efectos de la refactorización sobre atributos de calidad del software y mantener la consistencia entre los distintos artefactos software [1]. Desafortunadamente, las herramientas comerciales sólo asisten la aplicación automática de ciertas refactorizaciones. Además, las clasificaciones de refactorizaciones propuestas por diferentes autores no están orientadas a su implementación en herramientas. Para avanzar en

* Este trabajo ha sido subvencionado por el proyecto: Desarrollo, Integración y Evolución de Sistemas Software, un Enfoque de Líneas de producto. MEC-FEDER, TIN2004-03145.

el desarrollo de herramientas que asistan el proceso de refactorización completo, es necesario disponer de modelos que permitan caracterizar las refactorizaciones respecto al conjunto de actividades del proceso completo, así como de la complejidad intrínseca de su definición. A través de las distintas caracterizaciones, se debe dar soporte a la definición de criterios de selección de un conjunto de refactorizaciones y la determinación del orden de implementación, de manera que mejoren el proceso de desarrollo del producto. Se busca dar respuesta a preguntas del tipo: “Seleccionar un conjunto de refactorizaciones de diseño que asistan a la solución de un *bad smell* [2], indicando el orden de implementación de las mismas”

El resto del artículo se estructura de la siguiente forma, en la sección 2 se analiza un conjunto de trabajos donde aparecen diferentes clasificaciones de refactorizaciones. En la sección 3, se presenta un conjunto de criterios que permiten relacionar y unificar los diferentes conceptos asociados a las refactorizaciones. En la sección 4, se propone un modelo para almacenar los criterios establecidos y en la sección 5 se muestra una utilización de los mismos. Se termina con las conclusiones y líneas de trabajo futuro.

2. ANTECEDENTES

Los principales catálogos de refactorizaciones en los que se apoyan las herramientas son los expuestos por Opdyke [3] y Fowler [2]. Ambos autores, en la presentación de sus catálogos, muestran ciertos criterios de agrupación para mejorar la comprensión y la organización. Opdyke clasifica sus refactorizaciones en *Low Level Refactoring* y tres niveles de *High Level Refactoring*. Las 26 *Low Level Refactoring* están diseñadas para ser lo más simples posible y dar soporte en la definición de *High Level Refactoring*. Fowler, apoyado en los trabajos de Opdyke, presenta un catálogo de 68 refactorizaciones, organizado por un criterio funcional. Ambas clasificaciones se mantienen independientes con pocas relaciones entre ellas, incluso con solapamientos en algunas refactorizaciones de sus catálogos.

Por otro lado Fowler introduce el concepto de *Bad Smell* o defecto de código para iniciar el proceso de refactorización. Identifica 22 defectos relacionándolos con subconjuntos de refactorizaciones de su catálogo. En este sentido en [4], se amplía el trabajo de Fowler.

Basados en la alta complejidad de este dominio, otros trabajos se enfrentan al problema de clasificación de refactorizaciones y estructuración de su conocimiento. Su objetivo es mejorar la comprensión, unificar conocimiento y enseñar las relaciones existentes entre los diferentes elementos del dominio. En [5] se establecen siete clasificaciones para caracterizar las refactorizaciones aunque se centran fundamentalmente en el problema de mejorar la comprensión del conocimiento del dominio. En [6] se muestra una ontología que estructura y unifica el conocimiento acumulado en diseño en microarquitecturas orientadas a objeto. Identifica un conjunto de elementos del conocimiento entre los que se encuentran principios, heurísticas, mejores prácticas, *bad smells*, refactoring y patrones. Además se establece una jerarquía de clasificación de los mismos con sus relaciones.

Ninguno de los trabajos mencionados, se centra en una caracterización genérica y abierta de refactorizaciones, que asista a su implementación.

3. CARACTERIZACIÓN DE REFACTORIZACIONES

En este apartado se presenta un conjunto de criterios para caracterizar refactorizaciones de múltiples catálogos. Su aplicación individual, o la combinación de los mismos, debe guiar la implementación de refactorizaciones. Los criterios han sido agrupados para representar las relaciones entre refactorizaciones, el ámbito sobre el que actúa la refactorización, y el tipo de conocimiento asociado a su comprensión.

Los criterios son lo suficientemente generales para ser extendido a otros catálogos, en algunos casos definidos sobre entidades. Por ejemplo, si las refactorizaciones actúan sobre código [2] entonces las entidades serían: sistema, clases, atributos, métodos, parámetros, instrucciones, variables locales. Si se aplican sobre refactorizaciones de diagramas de estado [7], las entidades serían: acción, estado y transición.

3.1. Relaciones entre refactorizaciones

Se han identificado tres tipos de relaciones entre las refactorizaciones: similitud, uso y de composición.

-*Relación de similitud*: indica que los mecanismos de definición de la refactorización son muy similares en ambas refactorizaciones. Por ejemplo, en los mecanismos de la definición de *AddParameter* se advierte de la similitud con *RenameParameter*.

-*Relación de uso*: indica que en la definición de una refactorización puede ser necesaria la aplicación de otras refactorizaciones. Por ejemplo, la refactorización *ExtractClass* usa las refactorizaciones *ExtractField* y *ExtractMethod*.

-*Relación de composición*: indica el conjunto de operaciones atómicas que se incluyen en una refactorización. Por operaciones atómicas se considera el conjunto de transformaciones básicas sobre cualquiera de las entidades del sistema: *crear una entidad*, *actualizar una entidad* y *eliminar una entidad*. La definición de cualquier refactorización se puede hacer considerando una composición de este conjunto de transformaciones básicas. De esta forma, la refactorización *Extract Class* del catálogo de Fowler, puede ser vista como una composición de las siguientes transformaciones básicas: *crear clase*, *crear atributo*, *eliminar atributo*, *crear método*, *eliminar método*, *crear instrucción*, *eliminar instrucción*, *crear parámetro*, *eliminar parámetro*, *crear variable local* y *eliminar variable local*.

3.2. Ámbito de las refactorizaciones

El ámbito de una refactorización identifica y cuantifica el conjunto de entidades sobre las que actúa una refactorización. Se especifica como un conjunto de pares (multiplicidad, entidad), entendiendo por multiplicidad un subconjunto posiblemente infinito de los enteros no negativos.

Basado en las definiciones semiformales de Opdyke, el ámbito puede ser analizado desde diferentes perspectivas: entrada, precondition y acción (operación).

-*Ámbito de entrada*: identifica y cuantifica las entidades necesarias como entrada para efectuar una refactorización. Este criterio se muestra especialmente útil para poder

caracterizar y reutilizar los interfaces gráficos necesarios en una refactorización. Por ejemplo, las refactorizaciones *AddParameter* y *RemoveParameter* tienen como ámbito de entrada $\{(1, \text{método}), (n, \text{parámetro})\}$.

-*Ámbito de las precondiciones*: identifica y cuantifica las consultas sobre las entidades de un sistema previas a la aplicación de una refactorización. Las precondiciones definen el conjunto de condiciones que debe cumplir el sistema para poder llevar a cabo una transformación. Estas condiciones se basan en consultas sobre el estado del sistema y pueden ser compartidas entre refactorizaciones. Por ejemplo, el siguiente subconjunto de refactorizaciones tiene el mismo ámbito de sus precondiciones $\{(n, \text{clase}), (n, \text{método}), (n, \text{entidad})\}$: *Add Parameter*, *Form Template Method*, *Introduce Parameter Object*, *Remove Parameter*, *Replace Parameter with Method*.

-*Ámbito de las acciones*: identifica y cuantifica las operaciones atómicas mostrando la incidencia y alcance de la transformación. Junto con el criterio relación de composición definido puede asistir a la reutilización de las acciones de transformación asociadas a las refactorizaciones.

3.3. Tipos de conocimiento

Para que las herramientas asistan el proceso de refactorización deben tener en cuenta la relación entre el tipo de conocimiento declarativo (principios, heurísticas, *bad smells*) y las refactorizaciones. Desde el punto de vista de implementación de las herramientas se puede asistir la selección de un conjunto de refactorizaciones que actúen juntas. Por ejemplo, una herramienta que quiera eliminar código duplicado deberá incluir al menos las refactorizaciones *Extract Method*, *Extract Class*, *Pull Up Method* y *Form Template Method*.

Los requisitos de conocimiento hacen referencia a los conceptos básicos y avanzados asociados a la aplicación de la refactorización. En el caso de la orientación a objetos, bajo un criterio subjetivo y dejando abierta esta enumeración, se pueden considerar conocimientos avanzados: reflectividad, patrones de diseño, aserciones, etc.

4. MODELO DE CARACTERIZACIÓN

En este apartado se presenta un modelo de caracterización que posibilita el almacenamiento de los criterios. Se describen las diferentes entidades y relaciones identificadas en la Figura 1. Los distintos ámbitos sobre los que actúa una refactorización, se representan a través de las relaciones de composición existentes entre la clase *Refactoring* y la clasificación de ámbitos (*PreConditionScope*, *InputScope*, *ActionScope*). El ámbito (*Scope*) identifica y cuantifica las entidades mediante la clase de asociación *MultiplicityElement* que contiene la multiplicidad (*MultiplicityElement.multiplicity*) asociada a las diferentes entidades recogidas en la clase *Entity*.

Las relaciones de uso y similitud se representan a través de las relaciones reflexivas definidas en la clase *Refactoring* e identificadas respectivamente como *use* y *similarity*. Respecto a las relaciones de composición de transformaciones atómicas asociadas a una refactorización, se obtienen navegando por las relaciones *action* y *composeOf*.

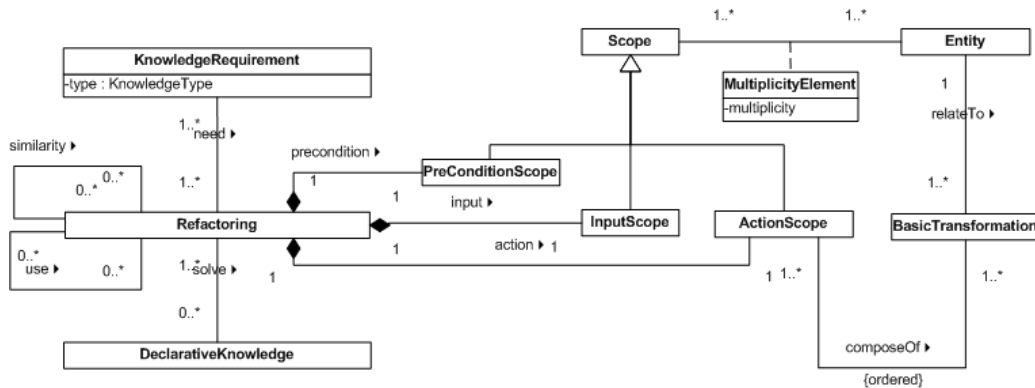


Figura 1 Modelos de caracterización de refactorizaciones

5. CASO DE ESTUDIO

En el caso de estudio propuesto se han caracterizado 44 refactorizaciones del catálogo de Fowler. En concreto las que se encuentran dentro de las categorías *Making Methods Calls Simpler*(15), *Dealing with Generalization*(12), *Moving features between Objects* (8) y *Composing Methods* (9). Las entidades asociadas han sido *System*, *Class*, *Method*, *Instruction*, *Parameter* y *Local Variable*. El único conocimiento declarativo recogido ha sido el relacionado con los *bad smells*. Las transformaciones básicas consideradas han sido *crear*, *actualizar* y *eliminar* sobre cada una de las entidades consideradas.

La definición de los criterios complejos se basa en aplicar operaciones de consulta sobre las clases del modelo, donde el campo *output* representa el conjunto resultado al aplicar una consulta o filtro.

5.1. Ordenar la implementación de refactorizaciones

En este caso de estudio se selecciona un conjunto de refactorizaciones de diseño que asisten la solución del *bad smell Large Class*, indicando el orden de implementación de las mismas.

Se describen los resultados obtenidos a la hora de aplicar cada paso junto con operaciones necesarias para definir los criterios:

- Seleccionar el subconjunto de refactorizaciones asociadas al *bad smell Large Class*:
output *Extract Class, Extract Subclass, Extract Interface.*
- Seleccionar aquellas que no trabajan con instrucciones en el ámbito de entrada:
output *Extract Class, Extract Subclass, Extract Interface.*
- Incluir en el conjunto las refactorizaciones de diseño asociadas con la relación de uso:
output *Extract Class, Move Field, Parameterize Method, Extract Subclass, Rename Method, Push Down Method, Push Down Field, Move Method, Extract Interface*
- Cuantificar el número de operaciones atómicas basado en las instancias de la asociación *composeOf* y ordenar las refactorizaciones respecto a la cuantificación:
output *Rename Method (1), Move Field(3), Push Down Field(3), Extract Interface (3), Parameterize Method(3), Move Method(6), Push Down Method(6), Extract*

Class (10), Extract Subclass(10).

6. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo se ha definido un modelo abierto que permite caracterizar refactorizaciones de distintos catálogos. El coste y dificultad de la caracterización bajo este modelo, puede ser compensado al desarrollar herramientas que incluyan refactorizaciones. En concreto, puede avanzar en la mejora de calidad en el desarrollo de la herramienta dotando al proceso de las siguientes características: organización, comprensión, reutilización y estimación de tareas.

Dado al alto grado de flexibilidad a la hora de combinar los criterios expuestos, la Tabla 1 establece una relación entre las características y los criterios expuestos. Sin embargo, por el momento, no se especifica de forma concreta sus posibles combinaciones, ni la obligatoriedad de usar todas las características, como se ha podido ver en el caso de estudio previo.

<i>Criterios \ Características</i>	<i>Organización</i>	<i>Comprensión</i>	<i>Reutilización</i>	<i>Estimación de Tareas</i>
Similitud	✓	✓	✓	✓
Uso	✓	✓	✓	✓
Composición	✓	✓	✓	✓
Ámbito de entrada	✓		✓	✓
Ámbito de precondiciones	✓		✓	✓
Ámbito de acciones	✓		✓	✓
Conocimiento Declarativo	✓	✓		
Requisito de conocimiento	✓	✓		✓

Tabla 1 Relación entre criterios y características

Se abre una línea futura de trabajo para validar el modelo respecto a catálogos de refactorizaciones que trabajan sobre otras entidades, como catálogos de refactorizaciones en diseño [7].

REFERENCIAS

- [1] Mens, T. and Tourwé T. “*A survey of software refactoring*”. IEEE Trans. Softw. Eng., 30(2):126–139, 2004.
- [2] Fowler, M. Refactoring. “*Improving the Design of Existing Code*”. Addison-Wesley, 2000.
- [3] William F. Opdyke. “*Refactoring Object-Oriented Frameworks*”. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 1992.
- [4] William C. Wake. “*Refactoring Workbook*”. Addison-Wesley, 2003.
- [5] Crespo Y., Marqués, J.M. “*Definición de un Marco de Trabajo para el análisis de refactorizaciones de software*”. En actas JISBD 2001, pages 297-310, Noviembre 2001.
- [6] Garzas, J., Piattini, M. “*An Ontology for Microarchitectural Design Knowledge*”. IEEE Software, vol. 22, no. 2, pp. 28-33, Mar/Apr, 2005
- [7] Sunye, G., Pollet, D. and Jezequel J. “*Refactoring uml models.*” In Proceedings of UML 2001: 134–148, 2001.