

Definición de un Proceso para la Construcción de Refactorizaciones

Raúl Marticorena	Carlos López	Yania Crespo
Área de Lenguajes y Sistemas Informáticos	Área de Lenguajes y Sistemas Informáticos	Dept. de Informática Universidad de Valladolid
Universidad de Burgos rmartico@ubu.es	Universidad de Burgos clopezno@ubu.es	yania@infor.uva.es

Resumen

La actividad de refactorizar el código es hoy en día una de las tareas de la mayoría de procesos de desarrollo del software, especialmente relevante en metodologías ágiles. Sin embargo, la definición de dichas refactorizaciones, su construcción e integración en herramientas software, no han sido abordadas desde un punto de vista de proceso, en particular en relación a la preservación del comportamiento. En este trabajo se presenta una aproximación a este tema, desde un enfoque con y para reutilización, aplicable sobre una familia amplia de lenguajes orientados a objetos.

Palabras claves: proceso, refactorización, reutilización

1. Introducción

En la actualidad, refactorizar el código se está convirtiendo en una tarea habitual. Siempre desde un punto de vista de mejorar su comprensión, facilitar su mantenimiento y preservar el comportamiento observable inicial [1] [2]. Dentro del proceso de refactorización se han detectado distintas líneas abiertas de investigación, sin embargo el propio proceso de definición, elaboración y construcción de las refactorizaciones y herramientas que las soportan está abierto, existiendo cierto desacuerdo entre las principales corrientes. Este trabajo intenta mostrar un enfoque de descomposición de los elementos de las refactorizaciones, que

permita utilizarlos en una forma incremental e iterativa en su ensamblaje y posterior ejecución, con un cierto grado de independencia del lenguaje.

En lo que sigue, este trabajo se estructura de la siguiente forma: la Sec. 2 presenta el estado del arte en la definición de refactorizaciones; la Sec. 3 muestra la solución actual de implementación; la Sec. 4 identifica los actores, tareas y productos en el proceso; en la Sec. 5 se presenta una solución estática, mientras que en la Sec. 6 se plantea dicho proceso desde un punto de vista iterativo e incremental, mostrando finalmente nuestra propuesta. Se finaliza en la Sec. 7, con las conclusiones y líneas de trabajo futuras.

2. Estado del Arte

Partiendo de los trabajos previos en refactorización se pueden observar distintas líneas en la definición de refactorizaciones y los elementos que las componen. En [3] se hace especial hincapié en el uso de precondiciones y en la verificación de la semántica estática y dinámica [4] a través del establecimiento de dichas precondiciones. Sin embargo esta solución es difícil de aplicar ya que depende de una definición precisa a priori. Además no se consideran los puntos de variabilidad, al extenderse la solución a otros lenguajes.

La misma línea de trabajo se continúa en [5], con el concepto añadido de postcondiciones. Sus definiciones son semiformales, utilizando lógica de predicados de primer orden sobre los

elementos que constituyen un programa orientado a objetos, permitiendo su composición. Ambos enfoques son pesimistas en el sentido de que pueden impedir ejecutar refactorizaciones que podrían ser correctas.

En [1] se proponen definiciones más informales, dando una lista de elementos que completan la descripción de una refactorización como una receta de cocina: nombre, resumen, motivación, mecanismos y ejemplos.

La preservación del comportamiento se garantiza a través de la compilación (asegurando la semántica estática) y la ejecución de las pruebas (asegurando la semántica dinámica), sin aparente criterio uniforme respecto a cuando compilar / probar. En este catálogo, una refactorización no especifica de una manera formal o semiformal las precondiciones que pueden impedir dicha refactorización. En todo caso, se puede extraer en forma textual algunas de las condiciones, pero sujetas a interpretación subjetiva. Este enfoque es más optimista, enfrentándose posteriormente al problema de errores en compilación o fallos en la ejecución de las pruebas.

Todas las propuestas señaladas son abordadas para un único lenguaje. Por otro lado se proporcionan definiciones de refactorizaciones, pero no se establece claramente un proceso de definición, implementación y prueba de las mismas.

3. Marco de Trabajo

En trabajos previos [6] se propuso y utilizó una plantilla para la definición de refactorizaciones. Mientras que de las propuestas previas, se mantienen algunas definiciones de elementos de forma textual, se incorpora una definición semiformal con lógica de predicados de primer orden sobre elementos básicos de una refactorización (entradas, pre/postcondiciones y acciones). Esta estructura puede ser aplicada a la mayoría de catálogos de refactorizaciones [1] [3] mejorando y completando en la mayoría de los casos dichas definiciones.

El enfoque estático y cerrado para la definición de los elementos de una refactorización es complicado de llevar a cabo, más aún cuando

se considera la búsqueda de una solución flexible y reutilizable que se pueda llevar a cabo sobre varios lenguajes. Para resolver este problema se toma como base el lenguaje MOON¹ [7] para la representación de la información en lenguajes orientados a objetos fuertemente tipados. A partir de dicho lenguaje se definen los elementos identificados en las refactorizaciones: nombre, descripción, motivación, entrada, precondiciones, acciones y postcondiciones.

Dicha representación se almacena utilizando una solución basada en frameworks, aportando una extensión al framework para cada lenguaje concreto del que se quiere capturar su semántica particular. En [6] también se ha planteado una solución para un motor de refactorizaciones. El motor permite ejecutar las refactorizaciones sobre las abstracciones del código, MOON en nuestro caso, modificando su estado, pero con una cierta independencia del lenguaje. Aunque los elementos participantes están claramente identificados y se tiene un diseño e implementación que soporte la solución, queda por definir un proceso claro y concreto que permita construir refactorizaciones.

4. Elementos en el Proceso

En esta sección se recogen los elementos identificados dentro del proceso de construcción y ejecución de refactorizaciones. Para la descripción de los elementos se utiliza la notación *Software Process Engineering Metamodel Specification (SPEM)* [8] en su versión 2.0.

4.1. Roles

Analista de Refactorizaciones: define los elementos de una refactorización. Este rol suele ser asumido habitualmente por el constructor del entorno integrado de desarrollo o herramienta de refactorización. La definición se basa en sus conocimientos y experiencia sobre dicho lenguaje. En nuestro caso particular debe tener conocimientos en cuanto a MOON y proyectar sobre él todos los elementos posibles del lenguaje particular como extensión.

¹MOON es acrónimo de Minimal Object-Oriented Notation

Ensamblador de Refactorizaciones: busca, construye y ensambla los elementos que componen una refactorización utilizando un repositorio. Este rol también es asumido habitualmente por el constructor del entorno integrado de desarrollo o herramienta de refactorización.

Usuario de Refactorizaciones: utiliza las refactorizaciones definidas sobre código escrito en un lenguaje de programación concreto. Cumple el papel de probar la corrección de las refactorizaciones asistido por herramientas externas como compiladores y entornos de automatización de pruebas.

4.2. Tareas

Análisis: estudio de la refactorización aplicada al lenguaje concreto sobre el que se trabaja. Puede implicar también la actividad de revisión de la definición ante posibles fallos posteriores en su aplicación.

Implementación: construcción de la parte operativa que es capaz de ejecutar dicha refactorización sobre la representación de código elegida. Extendiendo los elementos básicos del motor: predicados, funciones y acciones.

Refactorización: se ejecuta la refactorización sobre casos concretos utilizando el motor de refactorizaciones. Como resultado de la ejecución de la refactorización se puede producir un rechazo preventivo (incumplimiento de precondiciones) o un rechazo posterior a su aplicación (incumplimiento de postcondiciones, fallo de compilación, error en las pruebas, etc.)

4.3. Productos

Plantilla de Refactorización: está formada por los elementos básicos de la definición: predicados/refactorizaciones básicas [3], receta [1], plantilla de elementos [6], etc.

Refactorización: implementación y/o ensamblaje de los elementos que constituyen una refactorización. Los elementos se pueden implementar de cero o tomar de un repositorio.

Código Fuente: código a ser refactorizado.

Batería de pruebas: conjunto de pruebas que chequean el correcto funcionamiento del

código entre sucesivas refactorizaciones.

Informe de errores: errores detallados resultado de la ejecución de una refactorización (incumplimiento de la semántica estática o dinámica).

5. Definición Estática

En esta sección se presenta una aproximación a las tareas de análisis y refactorización sobre la solución propuesta en MOON, aunque se puede aplicar particularmente sobre cualquier lenguaje. Por motivos de brevedad se omite la descripción del ensamblaje, correspondiente a la implementación de la refactorización usando elementos del repositorio, según determina en cada caso la tarea de análisis. Posteriormente, en la Tabla 1, se muestran las características básicas de ambos procesos.

5.1. Comprobación Semántica Basada en Contratos

Se parte de la estructura basada en contratos (pre y postcondiciones), donde se completa la estructura de la refactorización sobre los elementos del lenguaje MOON, para posteriormente completar la definición con los elementos propios del lenguaje concreto sobre el que se va a ejecutar la refactorización. Esta tarea de definición exige un conocimiento tanto de MOON, como del lenguaje concreto sobre el que se aplica la refactorización.

En la Figura 1 se muestra la tarea y pasos correspondientes a la ejecución de las refactorizaciones. Como productos de trabajo de entrada se tiene la refactorización implementada a partir de su definición, mientras que el código fuente actúa como producto de entrada y salida. Como posibles situaciones de fallo, se encuentran el incumplimiento de precondiciones, o bien una incorrecta ejecución de la refactorización con fallo en las postcondiciones. No se establece realimentación presuponiendo una definición correcta de la refactorización para el lenguaje sobre el que se aplica.

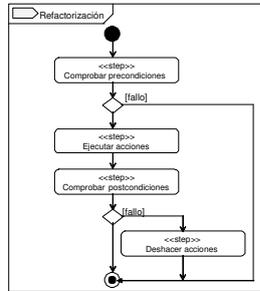


Figura 1: Ejecución de Refactorizaciones

5.2. Comprobación Semántica Basada en Compilación y Pruebas

Otra solución alternativa es no considerar el análisis y definición de precondiciones y postcondiciones a ningún nivel. En esta segunda solución, más próxima a lo postulado en [1], se toma un enfoque particular y dependiente del lenguaje, delegando las comprobaciones al compilador y batería de pruebas respecto a la corrección y preservación del comportamiento (ver Fig. 2).

La reutilización en el ensamblaje de la refactorización se limita al conjunto inicial de acciones definidas sobre MOON, que adicionalmente debe ser enriquecido con aquellas acciones propias definidas sobre el lenguaje concreto.

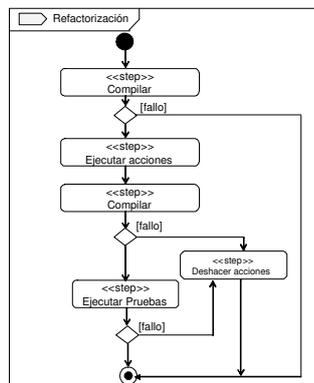


Figura 2: Ejecución de Refactorizaciones Basada en Compilación y Pruebas

En este caso también se parte de la suposición de que la definición inicial es correcta, particularmente de las acciones aplicadas al lenguaje concreto.

6. Proceso Iterativo e Incremental

A continuación, se presentan distintas evoluciones para establecer un proceso iterativo e incremental, que permita definir las refactorizaciones. Se concluye en la Tabla 1, con la comparativa entre las soluciones presentadas.

6.1. Comprobación Estricta de la Semántica

La combinación de ambas soluciones previas (Sec. 5.1 y Sec. 5.2), garantiza en mayor medida la preservación del comportamiento basada en las precondiciones (previo a la ejecución) y postcondiciones/compilación/pruebas (posterior a la ejecución).

En esta solución la definición de refactorizaciones se sigue realizando desde un punto de vista independiente del lenguaje, usando MOON, y el lenguaje concreto. Sin embargo la tarea de ejecución se enriquece con las fases de compilación y ejecución de la batería de pruebas (ver Fig. 3).

6.2. Comprobación Relajada de la Semántica

La complejidad de una definición correcta y completa de las precondiciones / postcondiciones de una refactorización, aplicada a un nuevo lenguaje concreto, suele derivar en un conjunto ineficaz o incorrecto de condiciones, que incluso desde el punto de vista de diferentes usuarios puede ser discutible.

En esta solución, se propone evitar dicho análisis de condiciones (pre / post) para el lenguaje concreto, reutilizando el análisis previo realizado sobre MOON, pero sólo incluyendo las acciones propias que puede requerir el nuevo lenguaje. La comprobación de la corrección de la refactorización, una vez ensamblada, se delegaría en el compilador (particular del

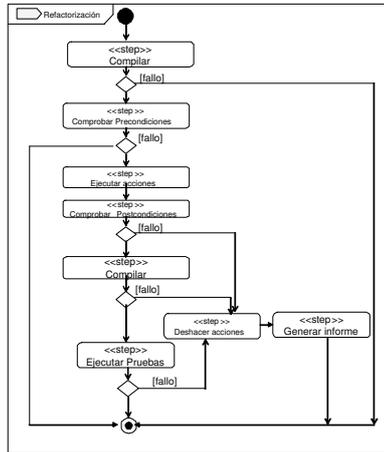


Figura 3: Ejecución de Refactorizaciones con Comprobación Estricta

lenguaje) y el juego de pruebas proporcionado.

6.3. Proceso Incremental de Definición

Se concluye proponiendo un proceso iterativo que incrementalmente complete las definiciones de las refactorizaciones, a medida que se obtienen resultados de una forma exploratoria. En esta solución, en la primera iteración no se establecen pre ni postcondiciones para el lenguaje concreto sobre el que se aplica la refactorización, pero sí para MOON. A continuación se va completando la definición en sucesivas iteraciones (ver Fig. 4) con las pre y postcondiciones particulares del lenguaje concreto. Sin embargo, este proceso incluye realimentación a partir de las ejecuciones de las refactorizaciones, con las definiciones establecidas.

Como se observa en la Fig. 5, una vez lanzado el informe de errores, éste pasa al analista, que puede revisar y definir de nuevo la refactorización (sus elementos), elaborando una nueva definición para que el ensamblador la implemente de nuevo con los elementos del repositorio.

Como resultado se obtiene una nueva im-

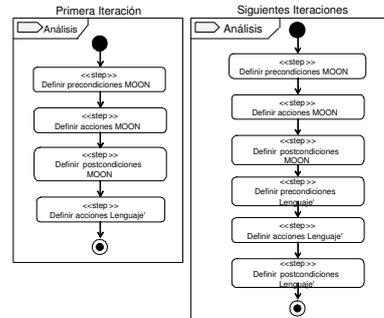


Figura 4: Definición Incremental de Refactorizaciones

plementación de la refactorización para que el usuario la utilice en sus tareas de refactorización. Ante fallos posteriores en la ejecución se repite el proceso.

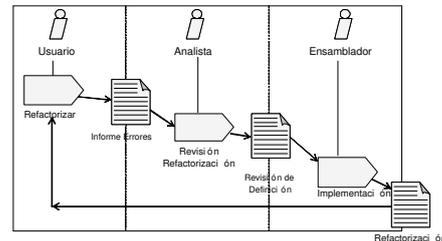


Figura 5: Diagrama de Actividad entre Actores

A medida que se observan rechazos por precondiciones muy restrictivas, o rechazos posteriores por estados incorrectos, se realiza una revisión de los elementos de la refactorización, desde un punto de vista de la independencia del lenguaje sobre MOON y desde un punto de vista del lenguaje particular sobre el que se trabaja, tal y como se propone en la Sec. 6.1.

Transcurrido un cierto número de iteraciones se debe llegar a un conjunto adecuado de pre y postcondiciones.

7. Conclusiones y Líneas de Trabajo

Como resultado de este trabajo se ha mostrado una solución de proceso que permite construir

Tabla 1: Comparativa de Procesos

Características positivas	Contratos (Sec.5.1)	Compilación y pruebas (Sec. 5.2)	Estricta (Sec. 6.1)	Relajada (Sec. 6.2)	Incremental (Sec. 6.3)
No necesita compilador	✓	-	-	-	-
No necesita juego de pruebas	✓	-	-	-	-
Rechazo por precondiciones	✓	-	✓	✓ (MOON)	✓ (MOON)
Asegura semántica estática	-	✓	✓	✓	✓
Asegura semántica dinámica	-	✓	✓	✓	✓
Sin coste pre / post	-	✓	-	-	-
Reutilización	Predicados Funciones Acciones	Acciones	Predicados Funciones Acciones	Predicados Funciones Acciones	Predicados Funciones Acciones
Pesimista/Optimista	P	O	P	P (parcial)	P (parcial - MOON)

refactorizaciones aprovechando los esfuerzos previos. En caso de necesitar aplicar estas refactorizaciones o sus elementos, sobre nuevos lenguajes, se define un proceso que a lo largo de varias iteraciones concluye con una implementación más adecuada. En la actualidad se trabaja sobre una definición declarativa de las refactorizaciones que permita su composición, a través de elementos del repositorio. Integrandolo su ejecución sobre el motor de métricas [6] y aplicando sobre dicho proceso el modelo de caracterización descrito en [9], se busca un proceso de definición y construcción de refactorizaciones, con y para reutilización con cierta independencia del lenguaje.

Referencias

- [1] Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Addison Wesley, 2000.
- [2] Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Trans. Softw. Eng.*, 30(2):126–139, 2004.
- [3] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 1992.
- [4] Bertrand Meyer. *Introduction to the Theory of Programming Languages*. Prentice Hall, 1990.
- [5] Donald Bradley Roberts. *Practical Analysis for Refactoring*. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 1999.
- [6] Yania Crespo, Carlos López, and Raúl Marticorena. Un framework para la reutilización de la definición de refactorizaciones. In *Actas JISBD'04, Málaga, Spain, ISBN 84-688-89830*, November 2004.
- [7] Yania Crespo. *Incremento del potencial de reutilización del software mediante refactorizaciones*. PhD thesis, Universidad de Valladolid, 2000. Available at <http://giro.infor.uva.es/docpub/crespo-phd.ps>.
- [8] *Software Process Engineering Meta-Model 2.0 OMG Draft Adopted Specification ad/2006-06-01*. June 2006.
- [9] Carlos López, Raul Marticorena, and Yania Crespo. Caracterización de refactorizaciones para la implementación en herramientas. In *XI Jornadas de Ingeniería del Software y Bases de Datos*, pages 538–543. ISBN:84-95999-99-4, Octubre 2006.