Extracción de clases

Identificación de oportunidades y refactorización automática

Carlos Jesús Peláez Rivas
Aspectos cualitativos y cuantitativos de la calidad del Software

Basado en:
Identification and application of Extract Class refactoring in object-oriented systems
Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, Alexander Chatzigeorgiou
University of Alberta (Canada), and University of Macedonia (Greece)

Introducción

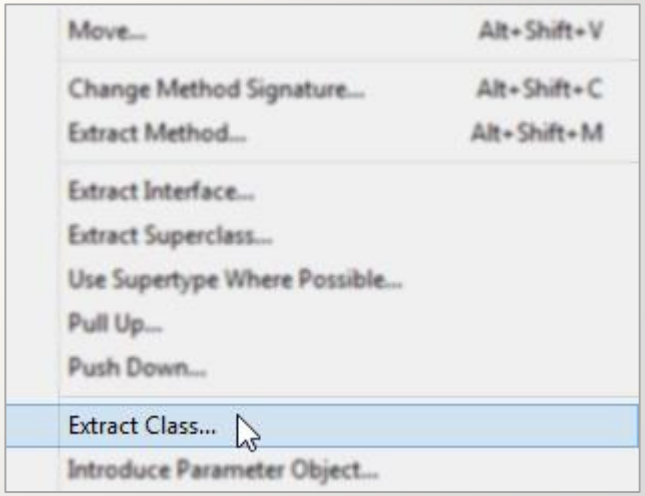
La refactorización es una práctica esencial en el desarrollo de productos Software.

El desarrollo ágil causa la introducción de *bad smells.*

La refactorización se vuelve necesaria.

Una de las refactorizaciones más conocidas es *Extract Class.*
Usada para mejorar clases grandes, complejas, pesadas e incoherentes, también conocidas como "God Classes".

Muchos IDEs incorporan ya esta herramienta.



Trabajo relacionado

El documento tiene una extensa bibliografía y trabajos relacionados divididos en 4 grupos:

- Métodos generales para la detección de *bad smells.*

  *U*sando diferentes técnicas y métricas.

- Investigaciones en remodularización de sistemas.

  Con la ayuda de algoritmos de *clustering* y optimización.

- Identificación de "*God Classes*".

  Técnicas para obtener un sistema de más calidad.

- Identificación de oportunidades para extraer clases.

  Uso grafos para evaluar semántica y cohesión.

Objetivos

El artículo estudiado tiene tres objetivos bien definidos:

a) **Reconocimiento de oportunidades para extraer clases.** *Detectando grupos de paquetes de datos coherentes o con comportamiento similar.*

b) **Puntuar y ordenar las oportunidades encontradas, en función de la mejora que aporta al diseño.** *Usando la métrica Entity Placement.*

c) **Refactorización automática del código usando la solución elegida por el desarrollador.** *Preservando la funcionalidad y mejorando el diseño.*

Identificación de clases a extraer

El proceso tiene dos pasos:

- Análisis de clases para extaer dependencias y cálculo de distancia entre sus miembros. Uso de un algoritmo que identifica los grupos coherentes.
- Las clases identificadas se filtran usando diferentes reglas para comprobar que tienen suficiente funcionalidad y mantienen el comportamiento.

Este proceso se aplica a todas las clases, sin comprobar antes si son 'God Classes' o no.

Identificación de clases a extraer

Algoritmo de agrupación

Ideas iniciales rechazadas:

- *Algoritmo k-medias*
- *Algoritmos basados en densidad, como DBSCAN*
- *Algoritmos de partición de grafos o basados en flujo*

Finalmente se usó un **algoritmo de aglomeración gerárquica** por los siguientes motivos:

- ✓ *Son deterministas*
- ✓ *Son finitos*
- ✓ *Son completamente automáticos. No requieren la definición de una entrada por parte del usuario.*

Identificación de clases a extraer

Cálculo de la distancia

Uno de los aspectos más importantes de los algoritmos de aglomeración gerárquica es el criterio de distancia usado para crear los grupos.

El criterio de distancia usado en este trabajo ha sido:

La distancia de Jaccard

$$d_{\alpha,\beta} = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Identificación de clases a extraer

Filtrado de casos identificados

La refactorización consiste en reestructurar el código sin que afecte a su comportamiento. Por ello, los casos detectados que cumplan alguna de las siguientes condiciones no serán válidos:

- Clases que no contengan métodos
- Clases con métodos abstractos
- Clases con campos cuya visibilidad sea superior a *private*
- Clases con métodos que sobreescriben a otro de su superclase
- Clases con métodos que contentan invocaciones a `super()`
- Clases que contentan métodos o bloques *synchronized*

Ordenación de las oportunidades

Una vez identificadas todas las oportunidades, se ordenan en función del impacto de su aplicación en la mejora del código.

Para ello se usa como métrica el valor:

Entity Placement

$$EP_C = \frac{\sum_{e_i \in C} distancia(e_i, C) \,/|entidades \in C|}{\sum_{e_i \notin C} distancia(e_i, C) \,/|entidades \notin C|}$$

Todas las posibles refactorizaciones son ordenadas de menor a mayor y mostradas al desarrollador.

Aplicación de refactorizaciones

Los autores crearon un plugin como extensión de **Jdeodorant Eclipse**, para ejecutar la refactorización elegida por el desarrollador.

Las acciones llevadas a cabo por el proceso son:

1. Eliminar las entidades a extraer de la clase original
2. Crear la nueva clase en el mismo paquete. Crear los *import* necesarios
3. Creación de los campos extraidos en la nueva clase. Se crean también los métodos *get* y *set* de dichos campos como métodos públicos
4. Añadir todos los métodos extraidos a la nueva clase (usando otro algoritmo para mantener el orden de los mismos)
5. Crear una referencia a la nueva clase en la clase original
6. Modificar los accesos en la clase original para usar la nueva referencia
7. Crear métodos *get* en la clase original si fueran necesarios
8. Cambiar la visibilidad de los métodos necesarios en la clase original

Evaluación del método

En el artículo se presentan tres tipos de evaluaciones diferentes:

- Evaluación de la precisión y el *recall*

  Comprobar si la metodología usada puede extraer nuevos conceptos embebidos en otras clases.

- Opinión del experto

  La evaluación por parte de un desarrollador de las refactorizaciones sugeridas.

- Comparación de métricas en un problema concreto

  Comprobar la validez de la métrica Entity Placement.

Evaluación del método

Precisión y *recall*

Se pidió a desarrolladores experimentados extraer clases manualmente en varios proyectos Software, para realizar una comparación con los resultados del método descrito.

Se contabilizaron los casos '*verdades positivos*', '*falsos positivos*', y los '*falsos negativos*'.

Resultados obtenidos (media de 3 proyectos):

67% de precisión, y 82% de *recall*

Evaluación del método

Medidas estructurales y semánticas

Hasta ahora se han usado medidas estructurales para realizar las agrupaciones.

Se compararon las medidas estructurales con las medidas semánticas, obteniendo resultados similares.

Los nombres dados a las variables y métodos son clave para que este tipo de medidas funcionen.

La combinación de ambas en una sola medida, perjudicó a los resultados. Se interferen mutuamente.

Evaluación del método

Opinión del experto

Se identificaron sobre un proyecto Software conocido diferentes opotunidades de extracción de clases.

Los resultados se mostraron a un desarrollador experimentado, y se le preguntó si dichas refactorizaciones eran útiles y si mejoraban la calidad del código.

En 9 de las 16 posibles refactorizaciones, el experto indicó que él realizaría dicha mejora.

En 3 de los casos el desarrollador indicó que no se había percatado de esa oportunidad de extracción.

Evaluación del método

Comparación de métricas en un problema dado

Comparación de las métricas para ver los efectos futuros de las refactorizaciones:

- *Entity Placement. Representa cohesión y acoplamiento.*
- *MPC (Message-Passing Coupling). Representa acoplamiento.*
- *Connectivity.* Representa cohesión.

El objetivo es reducir el acoplamiento, y aumentar la cohesión del sistema.

Las tres métricas muestran las tendencias esperadas, pero *Entity Placement* es suficiente para las evaluaciones necesarias en este método.

Conclusiones del trabajo

Los autores de este artículo han propuesto un método novedoso para mejorar el calidad de diseño en los sistemas orientados a objetos.

También desarrollaron una aplicación para ejecutar su solución de forma automática, una vez el desarrollador ha elegido la refactorización a ejecutar.

Las evaluaciones vistas, usando varios tipos de métricas, han tenido resultados positivos, y se ha demostrado que la métrica usada como ordenación de las refactorizaciones es buena para evaluar el impacto que tendrán los cambios en el diseño del sistema.

Trabajo futuro

- ✓ Mejorar el método de refactorización usado
- ✓ Combinar el algoritmo de agrupación para ofrecer resultados más completos
- ✓ Usar técnicas para detectar código duplicado
- ✓ Ofrecer refactorizaciones que solucionen varias oportunidades de extracción al mismo tiempo
- ✓ Mejorar el interfaz para mostrar usando un grafo las oportunidades de extracción de clases
- ✓ Ayudar al usuario a entender las refactorizaciones sugeridas

Valoración personal

Puntos positivos:

- ✓ El uso de esta herramienta podría ayudar a desarrollar Software más sencillo y coherente
- ✓ Contibuye a reducir las 'God Classes'
- ✓ El artículo está bien documentado y tiene bastante referencias, lo que indica un buen conocimiento del estado del arte en este campo
- ✓ Considero que todos los conceptos están bien explicados, aunque esto contribuye a un documento de mayor tamaño

Puntos negativos:

- ✘ No se incluye el nombre de la aplicación desarrollada, por lo que no se puede probar
- ✘ No se puede saber cómo reacciona la aplicación con un Software diferente al que han usado los autores
- ✘ El número de personas consultadas en las valoraciones ha sido muy bajo

Extracción de clases

Identificación de oportunidades y refactorización automática

Carlos Jesús Peláez Rivas
Aspectos cualitativos y cuantitativos de la calidad del Software

Basado en:
Identification and application of Extract Class refactoring in object-oriented systems
Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, Alexander Chatzigeorgiou
University of Alberta (Canada), and University of Macedonia (Greece)