УДК 621.518.54

Г.Г. Сергеев, канд. техн. наук
Севастопольский национальный технический университет
ул. Университетская 33, г. Севастополь, Украина, 99053
E-mail: kvt@sevgtu.sebastopol.ua

СТРУКТУРНЫЙ РЕФАКТОРИНГ КЛАССОВ ПРОГРАММНОГО ПРОДУКТА

Рассматривается формальная постановка задачи рефакторинга структуры классов объектно-ориентированных программных систем. Определено множество классов программного продукта, для которых целесообразно выполнение процедуры рефакторинга. Вводится в рассмотрение целевая функция, позволяющая представить задачу рефакторинга как задачу дискретной оптимизации. Приведены результаты экспериментов рефакторинга классов программного обеспечения диагностического комплекса «МАРС».

Ключевые слова: *рефакторинг, класс, объект, поле, метод, тип, дерево иерархии классов, агрегат, символьная запись.*

Процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы является основной задачей рефакторинга [1]. На практике, рефакторинг кода осуществляется разработчиками после разработки нового программного продукта или во время разработки нового продукта на основе существующего кода. Код нужно подвергнуть рефакторингу, если имеет место дублирование текста, длинные методы, большие классы, длинные списки параметров, чрезмерное обращение к данным другого объекта, избыточные временные переменные и.т.п.. Не смотря на то, что вопросам рефакторинга посвящен ряд фундаментальных работ, в том числе [1-2], большинство из них представляет собой совокупность рекомендаций, вследствие того, что понятие «читаемости» или «чистоты» кода невозможно формально определить. В работе [3] сделана попытка формализации процесса рефакторинга на основе определения символьной записи структуры классов и введения операций пересечения и вычитания.

Целью данной работы является рассмотрение слабо формализованного процесса рефакторинга как задачи оптимизации целевой функции. Для этого необходимо определить множество классов существующего программного продукта, над которыми целесообразно проведение структурной оптимизации, а также ввести критерии эффективности принимаемых решений.

Анализ современных объектно-ориентированных систем программирования показывает, что исходный код конечного программного продукта есть агрегация экземпляров объектов. Тип объекта определяется одним из неабстрактных классов в дереве иерархии $T=\{C_1, C_2, ..., C_n\}$. При этом в дерево T входит полное множество классов проекта, включая библиотечные пакеты.

В статье [3] был предложен способ символьной записи полей f^{type} и методов $m^{type}_{prim}(param)$ класса C Введем в рассмотрение отношение вида $f^{type} \mapsto C_i, m^{type} \mapsto C_i$. Это означает, что поле или метод имеют тип C_i или наследуемый от класса C_i. Тогда простое множества агрегируемых классов $C_i^S \subseteq T$ для класса C_i есть

$$C_i^S = \bigcup_{k=1}^{n} \left(C_k \left| \exists \left(\left(f^{type} \mapsto C_k \right) \vee \left(m^{type} \mapsto C_k \right), f^{type} \in F_i, m^{type} \in M_i \right) \right. \right).$$

Расширенное множество агрегируемых классов $C_i^E \subseteq T$ есть

$$C_i^E = \bigcup_{k=1}^{n} \left(C_k \left| \exists \left(\begin{array}{l} \left(f^{type} \mapsto C_k \right) \vee \left(m^{type} \mapsto C_k \right), \\ f^{type} \in \bigcup_r F_r, C_r \in C_i^S \bigcup C_i^E, \; m^{type} \in \bigcup_r M_r, C_r \in C_i^S \bigcup C_i^E \end{array} \right) \right. \right).$$

Другими словами, класс C_k входит в простое множество агрегируемых классов C_i^S, если среди типов полей и методов C_i есть тип C_k, или наследуемый от C_k. Не исключено, что $i=k$, то есть класс агрегирует объект такого же типа. Класс C_k входит в расширенное множество агрегируемых классов C_i^E, если среди объединения множества типов полей или методов C_i^S, а также классов уже вошедших в C_i^E есть поля или методы типа C_k. Очевидно, что процедура формирования множества C_i^E рекурсивна.

Под полным множеством агрегируемых классов $C_i^X \subseteq T$ будем понимать

$$C_i^X = \bigcup_{k=1}^{n} C_k \left| \left(\left(C_k \in C_i^E \right) \vee \left(\psi \mapsto C_k \right) \right) \right.,$$

где ψ — множество локальных переменных методов классов, входящих в C_i^E.

В исходном коде программного продукта имеется класс C_m, в состав которого входит метод main(), с которого начинается исполнение программы. Процедура рефакторинга $R(C^z)$, приводящая структуру классов множества C^z к каноническому виду за счет применения операций пересечения и вычитания символьных структур была рассмотрена в [3], однако какие классы должны войти в C^z не уточнялось. Очевидно, что для небольших проектов процедуру следует производить на всем дереве иерархии $R(T)$, а при больших объемах исходного кода — на множестве классов C_m^X. Дерево классов T можно считать функционально избыточным, если выполняется условие $T \setminus C_m^X \neq \varnothing$. Применение процедуры $R(C^z)$ к множеству классов C^z гарантирует, что

$$\bigvee_{i \neq j} \left(C_i \cap C_j \right) = \varnothing, \; C_i \in C^z, \; C_j \in C^z.$$

Для формальной оценки эффективности процедуры рефакторинга структуры классов можно предложить единый критерий на основе следующих рассуждений.

Пусть $C_i \in T$ и некоторый объект $\alpha \mapsto C_i$. Введем в рассмотрение коэффициент использования объектом ресурсов класса

$$\eta(\alpha, C) = \frac{|\overline{\alpha}|}{|C|},$$

где $|C| = |F| + |M|$ — алгебраическая сумма числа полей и методов класса. Величина $\overline{\alpha}$ — это множество реально используемых полей и методов класса C объектом α. Определим целевую функцию структурного рефакторинга

$$\min_{i,j} \eta(\alpha_i, C_j) \to \max. \qquad (1)$$

Для оценки $|\overline{\alpha}|$ необходимо ввести понятие области видимости объекта. Пусть P — цепочка лексем, из которых состоит код проекта. Область видимости $V(\alpha \mapsto C) \subseteq P$ — это совокупность цепочек, в которых имеется возможность использования публичных полей и методов объекта α.

В большинстве объектно-ориентированных языков для локальных объектов область видимости простирается от момента объявления объекта до конца программного блока, в котором он был объявлен. Для глобальных объектов область видимости распространяется на пакет и пакеты, использующие пакет в котором он был объявлен. Из области видимости следует исключить цепочки, в которых используются одноименные локальные объекты (рисунок 1).

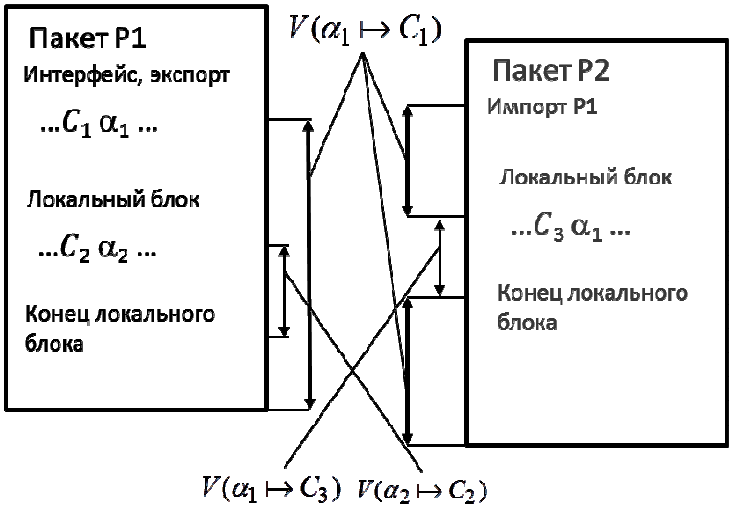

Рисунок 1 — Области видимости объектов программы

Чтобы определить мощность множества реально используемых полей и методов класса объекта введем понятие явно и косвенно используемых ресурсов. Определим множество явного использования ресурсов $\bar{\alpha}'$ как

$$\bar{\alpha}' = \left(\bigcup_k f_k, f_k \in C \middle| \exists(\alpha . f_k \in V(\alpha \mapsto C)) \right) \cup \left(\bigcup_l m_l, m_l \in C \middle| \exists(\alpha . m_l \in V(\alpha \mapsto C)) \right).$$

То есть $\bar{\alpha}'$ — это совокупность полей и методов объекта α, обращение к которым встречается в области видимости объекта.

Поле $f_k \in C$ является ресурсом метода $m_l \in C$, если лексема f_k встречается в тексте метода m_l. Отношение «является ресурсом» будем обозначать $m_l \Rightarrow f_k$. Метод $m_k \in C$ является ресурсом метода $m_l \in C$, если лексема m_k встречается в тексте метода m_l ($m_l \Rightarrow m_k$). Введем в рассмотрение матрицу инцидентности *MiR*, в которой строкам соответствуют методы, а столбцам — поля класса C

$$MiR[i, j] = \begin{cases} 1 \middle| \ m_i \in C, (m_i \Rightarrow f_j) \\ 0 \end{cases}$$

и матрицу смежности методов *MsM*, в которой строкам и столбцам соответствуют методы класса C

$$MsM[i, j] = \begin{cases} 1 \middle| \ m_i \in C, (m_i \Rightarrow m_j) \\ 0 \end{cases}.$$

На основании матрицы *MiR* можно построить двумерную матрицу смежности ресурсов *MsR*:

$$MsR[i, j] = \begin{cases} 1 \middle| \exists m_l \in C, \ MiR[l, i] \cdot MiR[l, j] \neq 0; \\ 0. \end{cases}$$

Тогда множество $\bar{\alpha}$ можно определить как

$$\bar{\alpha} = \left(\bigcup_k f_k \exists \left(\left(f_z \in \bar{a}' \right) \wedge (MsR[z, k] \neq 0), f_k \in C \right) \right) \cup \left(\bigcup_l m_l \exists \left(\left(m_z \in \bar{a}' \right) \wedge (MsM[z, l] \neq 0), m_l \in C \right) \right).$$

Таким образом, для обеспечения максимума целевой функции (1) необходимо исключить из класса C поля и методы, не принадлежащие $\bar{\alpha}$. То есть на всем множестве объектов $\alpha \mapsto C$ необходимо выполнить процедуру $R(\bar{\alpha})$. Оптимизация целевой функции (1) предполагает, что классы, представляющие объекты программы не содержат лишних переменных и связанных с ними методов. В результате компиляции достигается существенное сокращение сегмента кода и данных.

Обобщенная структура программного обеспечения, выполняющего автоматизированный рефакторинг структуры классов, показана на рисунке 2.

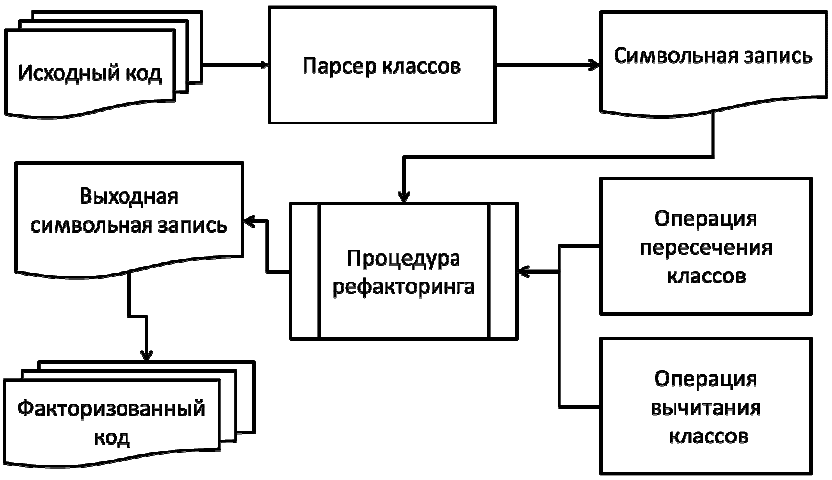

Рисунок 2 — Обобщенная структура процесса рефакторинга

Описанная выше методика рефакторинга существующего проекта была апробирована на примере классов программного обеспечения комплекса «МАРС» [4]. Анализ результатов экспериментов показывает, что рефакторинг классов программной системы приводит к росту объема исходного кода за

счет увеличения глубины дерева иерархии классов и роста числа абстрактных классов. Объем исполняемого файла остается практически неизменным. Не смотря на то, что цели рефакторинга и оптимизации взаимно противоречивы [1], следует отметить значительное (на 4 %) сокращение объема памяти, занимаемой программой в процессе исполнения на одних и тех же тестовых наборах.

В результате применения процедуры рефакторинга также достигается балансирование дерева иерархии. Это дает возможность упросить введение новой функциональности в унаследованных структурах. Статистические результаты экспериментов представлены в таблице 1.

Таблица 1 – Результаты рефакторинга классов ПО «MAPC»

Метрика ПО	До рефакторинга	После рефакторинга
Объем исходного кода проекта (строк)	65892	66498
Количество классов	216	251
Количество абстрактных классов	31	63
Целевая функция $\min\limits_{i,j}\eta(\alpha_i, C_j)$	0,51	0.87
Объем исполняемого файла (Мбайт)	1,71	1,67
Объем памяти в процессе исполнения (Мбайт)	67,35	63.21

В настоящее время разрабатывается новая методика, позволяющая выполнять не только структурный рефакторинг классов программной системы, но и фрагментов кода методов классов.

Библиографический список использованной литературы

1. Фаулер М. Рефакторинг: Улучшение существующего кода / М. Фаулер. — СПб.: Символ, 2003. — 432 с.

2. Ксензов М.В. Рефакторинг архитектуры программного обеспечения / М.В. Ксензов, Г.Г. Сергеев, В.М. Иванов // Труды Института системного программирования РАН. — М.: ИСП РАН, 2004. — Т. 8. — С. 180–192.

3. Сергеев Г.Г. Формализация процесса рефакторинга на основе символьной записи структуры классов / Г.Г. Сергеев // Вісник Національного технічного ун-ту «ХПІ». Тематичний випуск «Системний аналіз, управління та інформаційні технології». — Х.: НТУ «ХПІ», 2010. — № 67. — С. 100–104.

4. Сергеев Г.Г. MAPC – новый комплекс для диагностики и ремонта радиоэлектронного оборудования ВСВСУ // Сьома наукова конференція Харківського університету Повітряніх Сил імені Івана Кожедуба «Новітні технології – для захисту повітряного простору». — Х.: ХУПС ім. І. Кожедуба, 2011. — С. 79.

Поступила в редакцию 06.03.2012 г.

Сергєєв Г.Г. Структурний рефакторинг класів програмного продукту

Розглядається формальна постановка завдання рефакторингу структури класів об'єктно-орієнтованих програмних систем. Визначена безліч класів програмного продукту, для яких доцільне виконання процедури рефакторингу. Вводиться до розгляду цільова функція, що дозволяє представити завдання рефакторингу як завдання дискретної оптимізації. Наведено результати експериментів рефакторингу класів програмного забезпечення діагностичного комплексу «MAPC».

Ключові слова: рефакторинг, клас, об'єкт, поле, метод, тип, дерево ієрархії класів, агрегат, символьний запис.

Sergeev G.G. Structural refactoring of software classes

The formal task of structure of object classes refactoring is offered. The set of classes where the refactoring is necessary is defined. The problem of a refactoring is considered as a problem of discrete optimization. Criterion function is offered. Results of experiments of a refactoring of classes of the software of the diagnostic “MARS” complex are given.

Keywords: refactoring, class, object, field, method, type, tree of hierarchy of classes, unit, symbolical record.