УДК 519.68

Сергеев Г.Г.

РЕФАКТОРИНГ СТРУКТУРЫ ГРАФИЧЕСКИХ ОБЪЕКТОВ

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

The formal raising of task of refactoring structure of classes of object programmatic is offered. The great number of classes of refactoring software product is certain. The objective function of refactoring is offered. Refactoring task present as discrete optimization task. The results of experiments of refactoring classes in software of diagnostic complex «MARS» software are considered.

В настоящие время практически все программное обеспечение разрабатывается средствами объектно-ориентированного программирования. Именно это объясняет исключительную важность задач, связанных с облегчением сопровождения и развития существующего программного кода [1]. В то же время, ощущается явный недостаток методик и эффективных инструментов поддержки работы с существующим кодом. Процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы является основной задачей рефакторинга [2]. На практике, рефакторинг кода осуществляется разработчиками после разработки нового программного продукта ли 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.

Программное обеспечение систем с графическим интерфейсом, как правило, обладает наиболее сложным кодом с использованием большого количества уровней в дереве иерархии классов, абстрактных классов и интерфейсов. Символьная запись структуры программных классов рассматривалась в [3].

Целью настоящей работы является разработка методики, позволяющей сформировать рабочее множество классов, подлежащих рефакторингу и оценить влияние процесса рефакторинга на основные метрики программного обеспечения.

Согласно [3], Для описания структуры класса в виде символической записи предлагается следующая нотация. Символ, обозначающий атрибут класса, записывается как N^{type} , где *N* - имя поля, а *type* – тип поля. Метод класса предлагается записывать как $Meth_{prim}^{type,}(param)$, где *Meth* – идентификатор метода, *param* – перечень формальных параметров, а $prim \subseteq [virtual, abstarct, override]$. Допустимо, что $prim = \varnothing$ и/или $param = \varnothing$. Класс определяется как $Cname_{Cprim}$, где $Cprim = abstarct \left| \varnothing \right.$.

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.

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

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

Соответственно, 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. Обозначим процедуру рефакторинга, приводящую структуру классов множества C^z к каноническому виду за счет применения операций пересечения и вычитания символьных структур как $R(C^z)$. 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 имеет место 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, который может быть выражен как

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

Здесь: $|C| = |F| + |M|$, то есть алгебраическая сумма числа полей и методов класса. Величина $\overline{\alpha}$ – это множество реально используемых полей и методов класса C объектом α. Тогда целевую функцию структурного рефакторинга можно определить как

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

Для оценки $|\overline{\alpha}|$ необходимо ввести понятие области видимости объекта. Пусть P – цепочка лексем, из которых состоит код проекта. Тогда, 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.

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

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

То есть $\overline{\alpha}'$ – 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 $m_l \Rightarrow f_k$. Аналогично, 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

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

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

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

Тогда, множество $\overline{\alpha}$ можно определить как

$$\overline{\alpha} = \left(\bigcup_k f_k \exists \left(\left(f_z \in \overline{a}' \right) \wedge \left(MsR[z,k] \neq 0 \right), f_k \in C \right) \right) \cup$$

$$\cup \left(\bigcup_l m_l \exists \left(\left(m_z \in \overline{a}' \right) \wedge \left(MsM[z,l] \neq 0 \right), m_l \in C \right) \right).$$

Таким образом, для обеспечения максимума целевой функции (1) необходимо исключить из класса C поля и методы, не принадлежащие $\overline{\alpha}$. То есть на всем множестве объектов $\alpha \mapsto C$ необходимо выполнить процедуру $R(\overline{\alpha})$. Оптимизация целевой функции (1) предполагает, что классы, представляющие объекты программы не содержат лишних переменных и связанных с ними участков «мертвого» кода.

Рассмотрим пример рефакторинга программных классов, представленных на рисунке 1.

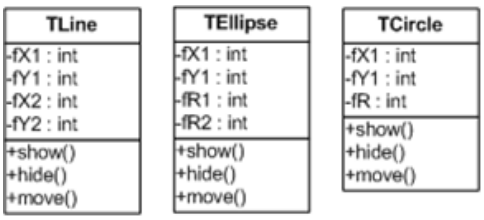

Рисунок 1 – UML – диаграмма классов простейших графических объектов

Символьная запись исходной структуры имеет вид:

$$TLine = TObject + \left\{ \begin{matrix} [fX1^{\text{int}}, fY1^{\text{int}}, fX2^{\text{int}}, fY2^{\text{int}}], [], \\ [show_{override}(), hide_{override}(), move_{override}()] \end{matrix} \right\},$$

$$TEllipse = TObject + \left\{ \begin{matrix} [fX1^{\text{int}}, fY1^{\text{int}}, fR1^{\text{int}}, fR2^{\text{int}}], [], \\ [show_{override}(), hide_{override}(), move_{override}()] \end{matrix} \right\},$$

$$TCircle = TObject + \left\{ \begin{matrix} [fX1^{\text{int}}, fY1^{\text{int}}, fR^{\text{int}}], [], \\ [show_{override}(), hide_{override}(), move_{override}()] \end{matrix} \right\}$$

В результате применения процедуры рефакторинга, описанной в [3] получим

$$TLine = TFigure + \{[fX2^{\text{int}}, fY2^{\text{int}}], [], [show_{override}(), hide_{override}()]\},$$

$$TFigure = BaseObject + \left\{ \begin{matrix} [fX1^{\text{int}}, fY1^{\text{int}}], [\ ], \\ [show_{virtua,abstractl}(), hide_{virtual,abstract}(), move()] \end{matrix} \right\},$$

$$TEllipse = TFigure + \{[fR1^{\text{int}}, fR2^{\text{int}}], [], [show_{override}(), hide_{override}()]\},$$

$$TCircle = TFigure + \{[fR^{\text{int}}], [], [show_{override}(), hide_{override}()]\}.$$

Описанная выше методика рефакторинга существующего проекта была апробирована на примере классов программного обеспечения комплекса «МАРС» [4]. Статистические результаты эксперимента приведены в таблице 1.

Таблица 1 – Результаты рефакторинга ПО «МАРС»

Метрика ПО	До рефакторинга	После рефакторинга
Объем исходного кода проекта (строк)	65892	66498
Количество классов	216	251
Количество абстрактных классов	31	63
Целевая функция $\min_{i,j} \eta(\alpha_i, C_j)$	0.51	0.87
Объем исполняемого файла (Мбайт)	1.71	1.67
Объем памяти в процессе исполнения (Мбайт)	67.35	63.21

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 в работе [1] отмечается, что цели рефакторинга и оптимизации взаимно протеворечивы, следует отметить значительное (на 4%) сокращение объема памяти, занимаемой программой в процессе исполнения на одних и тех же тестовых наборах.

Библиография

1. Фаулер М. Рефакторинг: Улучшение существующего кода / М. Фаулер. — СПб.:Символ, 2003. — 432 с.
2. Ксензов М.В. Рефакторинг архитектуры программного обеспечения / Г.Г. Сергеев, В.М. Иванов // Труды Института системного программирования РАН. — Т. 8. — М.: ИСП РАН, 2004. — С.180-192.
3. Сергеев Г.Г. Формализация процесса рефакторинга на основе символьной записи структуры классов / Г.Г. Сергеев // Вісник Національного технічного університету «ХПІ». . Тематичний випуск «Системний аналіз, управління та інформаційні технології» . — Х. : НТУ «ХПІ» . — 2010. — № 67. — С100-104.
4. Сергеев Г.Г. МАРС – 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.