СТРУКТУРНАЯ СЛОЖНОСТЬ UML-ДИАГРАММ КЛАССОВ

© 2016 г. О.А. ДЕРЮГИНА

Московский технологический университет (МИРЭА)
e-mail: o.a.derugina@yandex.ru

В процессе разработки программного обеспечения необходимо выполнять его рефакторинг — преобразование, сохраняющее поведение [1]; изменение внутренней структуры программного обеспечения без изменений его видимого поведения с целью облегчить понимание и удешевить модификации [2].

Целесообразно проводить рефакторинг в том числе и на этапе проектирования архитектуры программного обеспечения, к примеру, на основе UML-модели, т. к.: ·

- вносить изменения в модель менее трудозатратно, чем вносить изменения в уже готовый код;
- вероятность возникновения ошибок при внесении изменений на этапе проектирования также снижается.

Цель данной статьи — рассмотреть возможные критерии качества UML-диаграмм классов, которые могут быть использованы в качестве целевой функции для решения задач рефакторинга UML-диаграмм классов.

UML-диаграммы классов являются частью UML-модели, описывающей программную систему. UML-диаграммы классов описывают ПС с точки зрения её статических параметров: состав основных компонентов системы, связи между ними. UML-диаграммы классов описывают структуру ПС, её архитектуру.

Основным является вопрос о том, какие характеристики моделируемого программного обеспечения могут быть рассчитаны на основе UML-диаграммы классов.

В области качества программного обеспечения на территории РФ приняты следующие стандарты:

1.**ГОСТ 28806-90** – описывает модель качества программного продукта, а также даёт основные определения, связанные с качеством программного обеспечения.

2.**ГОСТ 28195-99** – описывает модель качества ПС, методы оценки качества, классифицирует методы измерения качества ПС.

Кроме того, разработана серия международных стандартов SQuaRE (Systems and Software Quality Requirements and Evaluation):

1.**ISO/IEC 2500n – Quality Management Division**. Определяет модели качества и определения, связанные с качеством программного обеспечения. Содержит требования и инструкции для поддержки управления требованиями, спецификации и оценки качества программного обеспечения.

2.**ISO/IEC 2501n – Quality Model Division**. Подробно описывает модели качества информационных систем и программного обеспечения, качества использования и данных.

3.**ISO/IEC 2502n – Quality Measurement Division**. Модель измерения качества продукта, математические определения метрик качества и практическое руководство по их применению.

4.**ISO/IEC 2503n – Quality Requirements Division**. Призван помочь в спецификации требований, основываясь на моделях качества и измерениях качества.

5.**ISO/IEC 2504n – Quality Evaluation Division**. Обеспечивает требования, рекомендации и руководство по оценке программного продукта.

6.**ISO/IEC 25050 – 25099 SQuaRE Extension Division**. Содержит требования к качеству коммерческого программного обеспечения и общих промышленных форматов для отчётов об удобстве использования.

Из предложенных в стандарте [3] критериев следующие могут быть применены к оценке UML диаграмм классов: модульность, модифицируемость, сопроводимость.

Модульность (modularity) – степень, с которой система или компьютерная программа составлена из дискретных компонентов так, что изменение одного компонента производит минимальное влияние на другие компоненты [3].

Модифицируемость (modifiability) – степень, с которой продукт или система может быть эффективно и производительно изменен без возникновения дефектов и без снижения текущего качества продукта [3].

Сопроводимость (maintainability) – степень эффективности и производительности, с которой продукт или система может быть изменён специалистами по сопровождению [3].

Исследователи предлагают различные метрики для количественной оценки UML-диаграмм классов. Наиболее распространёнными являются метрики Chidamber and Kemerer [4]:

1) Coupling Between Object (CBO – Связанность объектов) – число не связанных наследованием классов, с которыми связан данный класс. Отражает степень взаимосвязанности компонентов системы.

2) Response For a Class (RFC – Вызовы класса) – число методов класса и число вызываемых данными методами методов. Определяет степень сообщения между классами системы.

3) Lack of Cohesion in Methods (LCOM – Недостаток зацепления в методах) – число пар методов, которые не имеют общих переменных минус число пар методов, которые имеют разделяемые переменные класса.

4) Weighted Methods per Class (WMC – Взвешенные методы класса) – определяется как суммарная сложность методов класса. Если все величины сложности равны 1, то WMC равно число методов класса.

5) Depth of Inheritance Tree (DIT – Высота дерева наследования) – число уровней дерева наследования классов. DIT корневого класса дерева наследования равна 0.

6) Number of Children (NOC – Число потомков) – число непосредственных подклассов данного класса.

Однако оценить влияние данных метрик на модульность, модифицируемость и сопроводимость достаточно сложно.

Одной из целей рефакторинга объектно-ориентированной архитектуры ПС на основе UML-диаграмм классов может быть снижение структурной сложности UML-диаграмм классов (complexity).

Пусть дана диаграмма классов $d = \{C,I,R\}$, где C – множество классов $c_i \in d$, I – множество интерфейсов $i_i \in d$, R – множество отношений $r_i \in d$.

Тогда функция $K(d)$ измеряет структурную сложность диаграммы d следующим образом:

$$K(d) = |C| + |I| + |R| + \sum_{i=0}^{n} |A_i| + \sum_{i=0}^{n} |M_i| + \sum_{j=0}^{m} |M'_j|, \quad (1)$$

где $K(d)$ – структурная сложность диаграммы d; $|C|$ – количество классов $c_i \in d$; $|I|$ – количество интерфейсов $i_i \in d$; $|R|$ – количество отношений $r_i \in d$; A_i – множество атрибутов $a_j \in c_i$, $c_i \in d$; M_i – множество методов $m_i \in c_i$, $c_i \in d$; M'_j – множество методов $m_j \in i_i$, $i_i \in d$.

На основе предложенной в [5] абстрактной структуры данных UML Map данная метрика может быть рассчитана на языке Java следующим образом:

```
public int getComplexity(ClassDiagram classDiagram) {
       int c = classDiagram.getClasses().size(); //classes count
       int i = classDiagram.getInterfaces().size(); // interfaces count
       int r = classDiagram.getRelations().size(); // relations count
       //count attributes
```

```
        int a = 0; // attributes count
        for (Entry<String, Class> entry : classDiagram.getClasses().entrySet()) {
                Class c0 = entry.getValue();
                a += c0.getAttributes().size();
        }
        //count class methods
        int m1 = 0;
        for (Entry<String, Class> entry : classDiagram.getClasses().entrySet()) {
                Class c0 = entry.getValue();
                int m0 = c0.getOperations().size();
                int m00 = 0;
                //search interfaces, realized by this class
                for (Entry<String, Relation> relEntry : (classDiagram.getRelations()).entrySet()) {

                        Relation rel = relEntry.getValue();
                        if (rel.getStartId().equals(c0.getId())) {
                                if (rel.getType().equals("realization")) {
                                        m00 ++;
                                }
                        }
                }
                m1 += m0 - m00;
        }
        //count interface methods
        int m2 = 0;
        for (Entry<String, Interface> entry : classDiagram.getInterfaces().entrySet()) {
                Interface i0 = entry.getValue();
                m2 += i0.getOperations().size();
        }
        int complexity = c + i + r + a + m1+m2;
        return complexity;
}
```

В данной статье в качестве одного из возможных критериев качества для решения задачи рефакторинга UML-диаграмм классов предложена метрика структурной сложности.

Также приведён код на языке Java для расчёта данной метрики на основе абстрактной структуры данных UML Map.

СПИСОК ЛИТЕРАТУРЫ

1. *Kerievsky J.* Refactoring to Patterns // Boston M. A.: Addison-Wesley, — 2004.
2. *Fowler M.* Refactoring: Improving the Design of Existing Code // Boston M.A.: AddisonWesley — 2000.
3. ISO/IEC 2500n – Quality Management Division.
4. *Chidamber S.R., Kemerer C.F.* A metrics suite for object oriented design //IEEE Transactions on software engineering. — 1994. — Т. 20. — №. 6. — С. 476-493.
5. *Дерюгина О.А.* Трансформация UML-диаграмм классов с применением генетического алгоритма // Российский технологический журнал. — 2015. — Т. 1, № 4 (9). — С. 36-42.
6. *Дерюгина О.А.* Семантика и семантически эквивалентные трансформации UML-диаграмм классов // Труды МФТИ. — 2015. — Т. 7, № 2. — С. 146–155.