УДК 004.05

В.В. Бураков

ПОДХОД К МОДЕЛИРОВАНИЮ И ПОИСКУ ДЕФЕКТОВ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММНОГО КОДА

Бураков В.В. **Подход к моделированию и поиску дефектов объектно-ориентированного программного кода.**
Аннотация. В статье представлен подход к моделированию и идентификации дефектов программного кода для улучшения качества программного обеспечения. Основа подхода заключается в графовом моделировании исходного кода приложения и его комплексном анализе.
Ключевые слова: моделирование программ, графовая модель, дефект кода.

Burakov V.V. **An approach to model and identify object-oriented software code defects.**
Abstract. The paper presents an approach to model and identify defects in software code to improve their quality. The basis of the approach lies in the graph representation of the application source code and its complex analysis.
Keywords: software modeling, graph model, program defect.

1. Введение. В настоящее время при создании программного обеспечения бортовых комплексов широко используется объектно-ориентированное визуальное моделирование – молодая и бурно развивающаяся область компьютерной инженерии. В начале 90-х годов по этой теме появилось много фундаментальных работ. Наибольшее влияние на формирование этой области оказали исследования Г.Буча, И. Якобсона, Д. Рамбо, П.Коуда, Д.Харела, Б.Селика и др., усилиями которых был создан стандарт в этой отрасли – язык UML (UnifiedModelingLanguage) [6].

На сегодняшний день UML является широко используемым средством для проектирования программных комплексов любой сложности. Не смотря на это, UML обладает рядом существенных недостатков:

- неточная семантика. Так как UML определен комбинацией себя (абстрактный синтаксис), OCL (языком описания ограничений — формальной проверки правильности) и естественного английского (подробная семантика), то он лишен согласованности, присущей языкам, точно определенным техниками формального описания. В некоторых случаях абстрактный синтаксис UML, OCL и английский противоречат друг другу, в других случаях они неполные. Неточность описания самого UML одинаково отражается на пользователях и поставщиках инструментов, приводя к несовместимости инструментов из-за уникальной интерпретации спецификаций;

- UML не имеет свойств полноты по Тьюрингу и любой сгенерированный код будет ограничен тем, что может разглядеть или предположить интерпретирующий UML инструмент;
- кумулятивная нагрузка/рассогласование нагрузки. Как в любой системе обозначений – UML может представить одни системы более кратко и эффективно, чем другие. Таким образом, разработчик склоняется к решениям, которые более комфортно подходят к переплетению сильных сторон UML и языков программирования. Проблема становится более очевидной, если язык разработки не соответствует традиционным принципам разработки.

Исходя из вышеперечисленного, становится ясно, что концепции UML не позволяют смоделировать готовую версию программного продукта, из-за чего становится достаточно сложно выявить ошибки в архитектуре приложения на стадии проектирования. Помимо этого, при внесении изменений в архитектуру в процессе разработки, необходимо обновить построенные диаграммы классов и прочие схемы, что не всегда является удобным. В случае если разработчики этого не сделали, становится практически невозможным отследить возможные ошибки в архитектуре, так как средства обратной инженерии ограничены возможностями UML.

Эти факторы могут крайне негативно повлиять на надежность программного обеспечения, что является недопустимым в сферах, где от работы аппаратно-программного комплекса может зависеть здоровье и жизнь человека. Учитывая тенденцию всесторонней компьютеризации жизнедеятельности человека, надежность кода можно считать одним из основополагающих критериев качества. Из-за высокого объема программного кода наиболее эффективным способом выявления дефектов в системах подобного класса является автоматизированный поиск.

Важно иметь возможность строгого и согласованного в математическом смысле моделирования структуры и поведения разрабатываемых программных систем, а также разработать модели и алгоритмы для поиска дефектов архитектуры на любом этапе жизненного цикла программного обеспечения. Предлагаемый подход основан на использовании теории графов для представления кода и включает набор методов и алгоритмов для поиска дефектов в нем.

2. Графовая модель кода. Основой предлагаемой концепции является моделирование программного кода с помощью ориентированного помеченного типизированного графа. Помимо основного графа, представляющего исходный код программы, используется множество

подграфов, описывающих дефекты и условия появления этих дефектов. Для более полного описания условия наличия дефектов программного кода введены допустимые и недопустимые графы [1–5].

Определение 1. Ориентированный граф $G=(V,E,s,t)$ состоит из двух множеств, конечного множества V, элементы которого называются вершинами, и конечного множества E, элементы которого называются дугами. Каждая дуга связана с упорядоченной парой вершин. Для обозначения вершин используются символы $v_1,v_2,v_3...$, а для обозначения дуг — символы $e_1,e_2,e_3...$ Если $e_1=(v_i,v_j)$, то v_i и v_j называются оконечными вершинами e_1, при этом v_i — начальная вершина, а v_j — конечная вершина e_1. Все дуги, имеющие одну пару начальных и конечных вершин, называются параллельными.

Функции $s:E\to V$ и $t:E\to V$ связывают с каждой дугой в точности одну начальную и одну конечную вершины.

Определение 2. Помеченный граф. Пусть $L=(VL,EL)$, $A=(VA)$ — пара непересекающихся потенциально бесконечных множеств меток и ролей соответственно. (L,A) — помеченный граф G представляет собой тройку (g,l,a) такую, что имеет место:

1) $g=(V,E,s,t)$ — граф;

2) $l=(\mathrm{vl}:V\to VL,\mathrm{el}:E\to EL)$ — пара функций пометки соответственно вершин и дуг, при этом vl является инъективной;

3) $a=(\mathrm{va}:V\to VA)$ — функция отображения вершин на множество ролей.

Определение 3. Помеченный типизированный граф. Пусть $T=(VT,ET)$ — пара непересекающихся конечных множеств предопределенных типов вершин и дуг. (L,A)-помеченный T-типизированный граф G является двойкой (g,type) такой, что g является (L,A)-помеченным графом, а $\mathrm{type}=(\mathrm{vt}:V\to VT,\mathrm{et}:E\to ET)$ — пара функций, vt связывает с каждой вершиной из V ее тип из VT, а et — с каждой дугой из E его тип из ET.

Определение 4. Подграф. H является подграфом G (обозначается $H\to G$), если существует инъективный графовый морфизм $m:H\to G$, называемый соответствием H в G.

Определение 5. Допустимый (недопустимый) граф. Пусть $T = (VT, ET)$ — пара непересекающихся конечных множеств типов, а TT — T-помеченный граф, называемый допустимым (недопустимым) графом. (L, A) — помеченный TT-типизированный граф является двойкой (G, tt), такой, что G — (L, A)-помеченный граф и $\mathrm{tt} : G \to H$ — тотальный LGraph-морфизм. Помеченный TT-типизированный графовый морфизм представляет собой графовый морфизм вида $f : G \to H$ между (L, A) — помеченными TT-типизированными графами. TT-сохраняющий помеченный типизированный графовый морфизм представляет собой помеченный TT-типизированный графовый морфизм вида $f : G \to H$, такой, что $\mathrm{tt}_H \circ f = \mathrm{tt}_G$ для $\forall x \subseteq \mathrm{dom}(f)$.

На рис. 1 показан пример представления исходного кода программы на языке C++ в виде графовой модели.

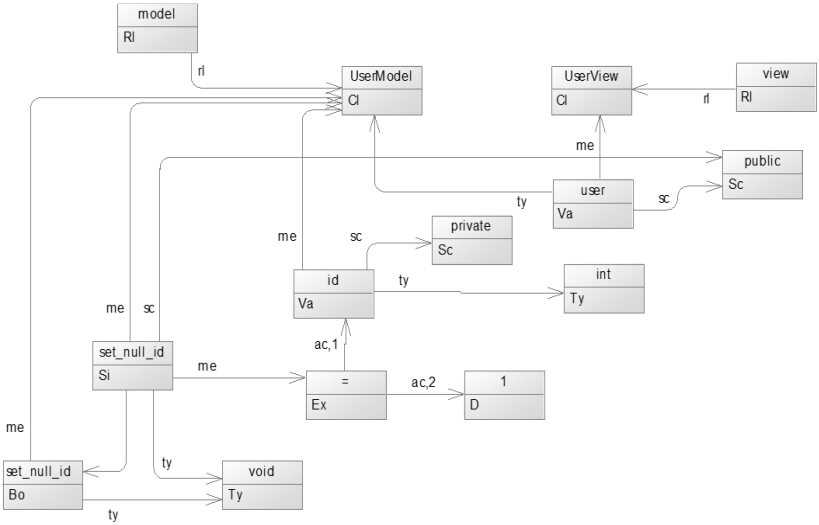

Рис. 1. Представление исходного кода в виде графовой модели.

Ниже представлен исходных код на языке c++ для данного графа:

```
   class UserView{//role=view
public: UserModel user;};
   class UserModel{//role=model
private:int id;
      public: void set_null_id(){id=0;}};
```

3. Спецификация и поиск дефектов. В основе описываемого подхода к поиску дефектов лежит представление программного кода и описания дефекта в виде графовой модели. В простейшем случае задача выявления дефекта сводится к поиску подграфа в графе. В терминах графовой модели дефектом может являться как факт нахождения подграфа в графе, так и отсутствие искомого подграфа.

Помимо поиска подграфа в графе, описываемый подход использует:

- распознавание дефектов на основе метрического анализа;
- роли для специализации контекста разрабатываемых программных сущностей и поиска дефектных программных решений;
- спецификации условий и поиск программных сущностей, нарушающих эти условия.

Конструктивно эти варианты поиска дефектов оформлены в виде плагинов, которые можно комбинировать в зависимости от конкретных задач анализа кода.

3.1. Метрический анализ. Суть метрического анализа заключается в подсчете количественных характеристик кода. По умолчанию определены семь базовых метрик:

- количество входных вершин вершины;
- количество выходных вершин вершины;
- количество входных вершин для вершины с дугами определенного типа;
- количество выходных вершин для вершины с дугами определенного типа;
- количество входных дуг определенного типа вершин для вершины;
- количество выходных дуг определенного типа вершин для вершины;
- длина нисходящего пути вершины из дуг определенного типа.

На основе базовых метрик пользователь может сформировать производные метрики. Каждая производная метрика определяется путем введения функциональной зависимости от n ($n>0$) других, как базовых, так и производных метрик. Основными видами функциональных зависимостей, порождающих производные метрики являются:

- сумма n базовых (производных) метрик;
- частное от деления одной базовой (производной) метрики на другую;

— максимальное значение базовой (производной) метрики;
— минимальное значение базовой (производной) метрики;
— среднее (арифметическое, геометрическое и т.п.) значение базовых (производных) метрик.

Для каждой метрики есть возможность задания граничных значений, что позволяет более гибко настраивать систему под различные проекты.

3.2. Роли и контекст. Каждому классу в проекте присваивается его роль (или роли), определяющие контекст использования экземпляров этого класса другими программными элементами. Также задаются правила отношений между классами с определенными ролями. В каждом правиле есть возможность указать как разрешенные связи, так и запрещенные. Основываясь на заданных правилах, для идентификации дефекта производится поиск запрещенных связей между сущностями программного кода. Например, пусть необходимо выявить корректность реализации архитектуры «Модель-представление-контроллер» (MVC). Суть подхода MVC заключается в отделении логики модели от логики представления и контроллера, таким образом, необходимо исключить прямые обращения из класса модели в классы представления и контроллера. На рис. 2 представлены графы, моделирующие эти дефекты. Суть идентификации дефекта заключается в нахождении этих подграфов в графе исходного кода.

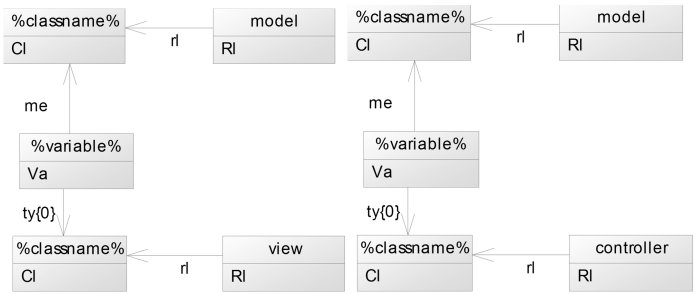

Рис. 2. Графовые модели дефектов архитектуры MVC.

3.3. Условия. В практике программирования относительно редко встречаются случаи, когда какая-либо часть программного кода является сама по себе дефектом. Как правило, для этого необходимо выполнение одного или нескольких условий. Причем даже для незначительного дефекта список условий может быть достаточно большим.

Каждый элемент списка условий представляет собой граф или набор графов, представляющих некое условие. Каждый элемент имеет

свой порядковый номер, необходимый для случаев, когда важен порядок выполнения условий. Если порядок условий важен только для нескольких элементов списка – порядковые номера задаются только для них. Помимо этого, каждый элемент снабжается признаком, содержащим в себе информацию о том, является данное условие обязательным или нет.

Граф, описывающий, некое условие также может быть допустимым или недопустимым. Для выявления некоторого дефекта необходимо, чтобы все обязательные условия были выполнены. Необязательные условия являются уточнениями к дефекту, так как не всегда могут существовать в исходном коде.

Допустим, нам необходимо в классах определенного типа выявить все методы, в которых вместо типа float используется тип double. Для начала нам необходимо описать простой дефект наличия переменной типа float. На рис. 3 описаны условия для этого дефекта. Первое условие является составным, то есть представляет собой набор условий, выполнение одного из которых (в данном случае), говорит о выполнении основного условия. В первом условии необходимо выявить группу классов, в которых осуществляется поиск дефекта. Поиск происходит либо по присвоенной роли классу, либо по наличию в этом классе функции с определенным именем. Последнее условие необходимо для идентификации дефекта в функциях с возвращаемым типом double. Все условия для данного дефекта являются обязательным, и невыполнение какого либо из них говорит о том, что часть программного кода не может быть идентифицирована как дефект.

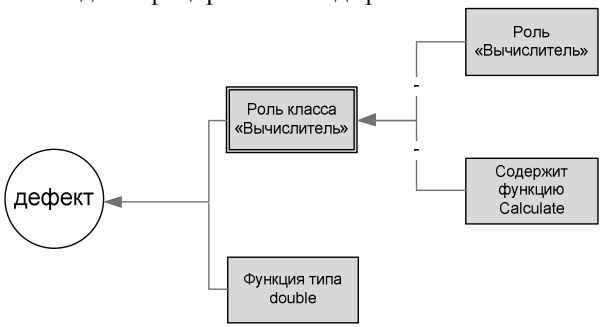

Рис. 3. Пример описания условий для дефекта.

4. Практическая апробация. В рамках исследований, финансируемых грантом РФФИ № 13-08-01250, для практической оценки эффективности предлагаемого подхода была разработана программная

система. В качестве платформы для ее реализации была выбрана среда разработки Microsoft Visual Studio. Компоненты, реализующие вышеуказанные этапы, представляют собой набор программных модулей, включенных в плагин к этой среде (рис. 4).

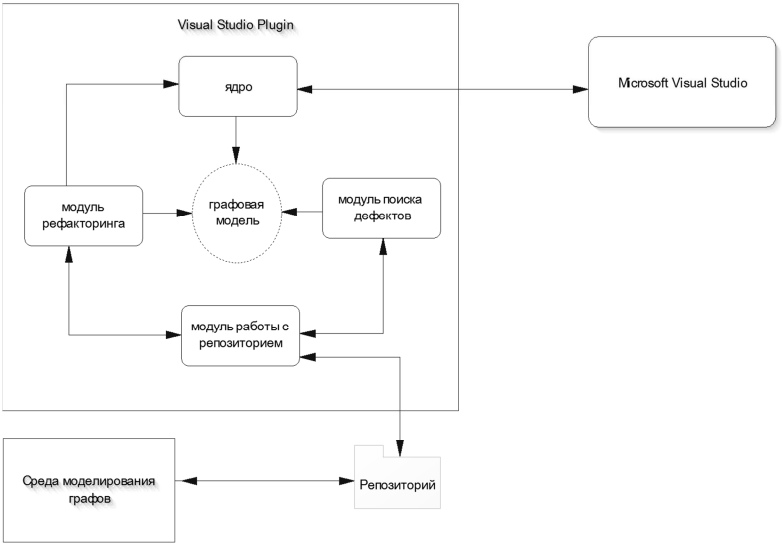

Рис. 4. Взаимодействие модулей плагина.

Среда взаимодействует непосредственно с модулем «ядро», который позволяет работать с объектной моделью открытого проекта. Большая часть модуля «ядро» представляет собой парсер, преобразующий код проекта в графовое представление. Непосредственно в сам плагин не входят репозиторий дефектов и рефакторигов и среда моделирования графов. Репозиторий представляет собой совокупность каталогов на диске, содержащих в себе XML файлы с представлением того или иного дефекта или рефакторинга, описанного в виде графа. Модуль поиска дефектов, получив описание дефектов из репозитория, производит поиск в графовой модели, сгенерированной при помощи ядра.

Основные возможности разработанной системы состоят в следующем:

- математически точное описание структуры и поведения классов;
- математическое моделирование дефектов;

- возможность ведения и пополнения базы дефектов;
- использование разных стратегий для поиска дефектов, выбор наиболее подходящей из них, возможность комбинирования стратегий;
- гибкая настройка системы поиска;
- автоматизированный поиск дефектов.

В настоящий момент осуществляется опытная эксплуатация разработанной системы моделирования и поиска программных дефектов, происходит формирование статистической информации, на основании которой можно будет адекватно оценить эффективность подхода.

5. Заключение. Предлагаемый подход позволит автоматизировать процессы выявления дефектов в архитектуре и структуре программных приложений, обеспечит повышение качества и снижение затрат на их разработку и сопровождение.

Литература

1. *Бураков В.В.* Концептуальное моделирование качества программных средств // Авиакосмическое приборостроение. 2008. № 7. С. 54–60.
2. *Бураков В.В.* Оценка качества программных средств // Авиакосмическое приборостроение. 2009. № 4. С. 28–33.
3. *Бураков В.В.* Система управления качеством программных средств // Авиакосмическое приборостроение. 2009. № 5. С. 27–31.
4. *Бураков В.В.* Управление качеством программных средств. СПб: ГУАП, 2009. 287 с.
5. *Бураков В.В.* Управление качеством программных средств // Информационно-управляющие системы. 2009. № 5. С. 43–47.
6. http://www.uml.org/

Бураков Вадим Витальевич — д.т.н., доц.; ведущий научный сотрудник лаборатории информационных технологий в системном анализе и моделировании СПИИРАН. Область научных интересов: программная инженерия, качество программных средств. Число научных публикаций — более 30. v.v.burakov@gmail.com, www.litsam.ru; СПИИРАН, 14-я линия, д. 39, Санкт-Петербург, 199178, РФ; р.т. +7(812) 328–0103, факс +7(812) 328–4450.

Burakov Vadim Vitalyevich — Ph.D., Dc.Sci., Assoc. Prof.; leading researcher, Laboratory for Information Technologies in Systems Analysis and Modeling, SPIIRAS. Research interests: software engineering, software quality. The number of publications — over 30. v.v.burakov@gmail.com,www.litsam.ru; SPIIRAS, 14th Line, 39, St.Petersburg, 199178, Russia; office phone +7(812) 328–0103, fax +7(812) 328–4450.

Поддержка исследований. Данная работа была выполнена при финансовой поддержке РФФИ, грант № 13-08-01250.

Рекомендовано лабораторией информационных технологий в системном анализе и моделировании, заместитель директора по научной работе Соколов Б.В., д.т.н., проф.
Статья поступила в редакцию 18.09.2013.

РЕФЕРАТ

Бураков В.В. **Подход к моделированию и поиску дефектов объектно-ориентированного программного кода.**

В статье предлагается новый подход, ориентированный на повышение качества объектно-ориентированного программного кода. Описаны процессы спецификации и поиска программных дефектов. Основой предлагаемой концепции является моделирование программного кода с помощью ориентированного помеченного типизированного графа. Спецификация программных дефектов осуществляется с помощью метрик, ролей и условий. Суть метрического анализа заключается в подсчете количественных характеристик кода. По умолчанию определены семь базовых метрик, на их основе можно формировать производные. Для каждой метрики есть возможность задания граничных значений, что позволяет более гибко настраивать систему под различные проекты. Помимо метрик, программные дефекты описываются с помощью ролей и контекста их использования. Каждому классу в проекте присваиваются его роли, определяющие контекст использования экземпляров этого класса другими программными элементами, задаются правила отношений между классами с определенными ролями. В каждом правиле есть возможность указать как разрешенные связи, так и запрещенные. Основываясь на заданных правилах, для идентификации дефекта производится поиск запрещенных связей между сущностями программного кода. Помимо модели исходного кода программы, графы также используются для описания дефектов. Для поиска дефектов осуществляется поиск подграфов, описывающих дефекты в графе, моделирующем программную систему. Предлагаемый подход позволит автоматизировать процессы выявления дефектов в архитектуре и структуре программных приложений, обеспечит повышение качества и снижение затрат на их разработку и сопровождение.

SUMMARY

Burakov V.V. **An approach to model and identify object-oriented software code defects.**

In article the new approach focused on improvement of quality of an object-oriented program code is offered. Processes of the specification and search of program defects are described. Basis of the offered concept is modeling of a program code by labeled typed digraphs. The specification of program defects is carried out by means of metrics, roles and conditions. The essence of the metric analysis consists in calculation of quantitative characteristics of a code. Seven basic metrics are by default defined, on their basis it is possible to form derivatives. For each metrics there is a possibility of a task of boundary values that allows to adjust more flexibly system under various projects. Besides metrics, program defects are described by means of roles and a context of their use. To each class in the project it is appropriated its roles defining a context of use of copies of this class by other program elements, rules of the relations between classes with certain roles are set. In each rule there is an opportunity to specify both the allowed communications, and forbidden. Based on the set rules, for identification of defect it is prospected the forbidden communications between entities of a program code. Besides model of an initial code of the program, the labeled typed digraphs also are used for the description of defects. Offered approach will allow to automate processes of detection of defects in architecture and structure of program appendices, will provide improvement of quality and decrease in expenses for their development and maintenance.