УДК 004.4(045)

П. К. Вытовтов, магистрант, ИжГТУ имени М. Т. Калашникова
Е. М. Марков, кандидат технических наук, ИжГТУ имени М. Т. Калашникова

РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ ПРОВЕДЕНИЯ СТАТИЧЕСКОГО АНАЛИЗА КОДА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Введение

Существует множество программных продуктов для проведения статического анализа программного кода. Среди наиболее распространенных решений можно выделить *PVS-Studio*, *Cppcheck*, *CppCat* и встроенный инструмент *Microsoft Visual Studio*. Основная их направленность – исправление синтаксических ошибок и ошибок набора. При этом существующие метрики кода программного обеспечения учитываются редко, несмотря на то, что их использование дает представление о том, где находятся «узкие» места анализируемых алгоритмов.

Вследствие указанной причины нами была поставлена задача разработать программный продукт для вычисления метрик программного кода и его комплексного показателя качества [1, 2]. В рамках данной работы был выполнен подробный анализ существующих метрик [3, 4] с выделением наиболее подходящих для поставленной задачи, определено направление разработки собственной комплексной меры [5] и выявлено, что промежуточное представление программного кода через абстрактное синтаксическое дерево [6] не является оптимальным – был выбран язык программирования *LLVM IR* [7].

1. Описание принципа работы программного продукта

Ключевой особенностью разрабатываемого решения является возможность снятия метрик кода программного обеспечения и вычисления комплексной обобщающей меры, которая в сочетании с остальными базовыми метриками предоставляет пользователю возможность получить представление об уровне качества программного кода анализируемого продукта.

Так как на различных этапах развития программного продукта требуется проводить анализ разных объемов программного кода, были предусмотрены три режима работы и соответствующие им модели поведения пользователя.

1.1. Анализ всего объема кода программного продукта

В данной модели поведения пользователь должен запустить программу и указать путь к каталогу проекта. Далее по нажатию соответствующей кнопки запускается трансляция исходного программного кода всего проекта в представление на промежуточном языке программирования (ЯП). Далее осуществляется снятие метрик с каждой отдельной функции программного продукта, дальнейшее их приведение к общему виду и отображение пользователю.

1.2. Анализ отдельного модуля программного продукта

В данном случае пользователь также должен запустить программу и указать путь к каталогу проекта. Далее следует выбрать отдельный файл, пользуясь деревом файлов проекта. По нажатию соответствующей кнопки запускается трансляция исходного программного кода выбранного файла в представление на промежуточном ЯП. Далее осуществляется снятие метрик с каждой функции выбранного программного модуля, их приведение к общему виду и отображение пользователю.

1.3. Анализ отдельной функции программного продукта

Здесь, как и в двух предыдущих моделях, после запуска программы пользователь должен указать путь к каталогу проекта. Далее ему следует выбрать некоторый файл проекта, и программа отобразит список функций данного модуля. После этого пользователь выбирает некоторую функцию и по нажатию соответствующей кнопки запускает трансляцию исходного программного кода выбранного модуля и снятие метрик кода программного обеспечения с выбранной функции. Здесь же пользователь может запросить отображение графа потока управления (ГПУ) выбранной функции.

2. Описание логики работы программного продукта

Исходя из описанных в разделе 1 моделей поведения условно можно выделить четыре основных задачи, решение которых требуется реализовать в логике работы разрабатываемого программного продукта.

2.1. Сканирование и отображение дерева файлов проекта

Решение данной задачи производится с помощью рекурсивного просмотра подкаталогов основной директории проекта. В момент выполнения данной работы проверяются расширения файлов и при их сов-

падении с допустимыми значениями запоминаются пути к файлам.

Далее нам требуется отобразить полученный список. Для этого мы используем элемент QTreeView из библиотеки *Qt*, для которой реализована модель представления данных на основе классов, отвечающих за хранение информации о подкаталогах и файлах проекта. По завершении формирования списка файлов проекта мы отображаем их в графическом интерфейсе.

2.2. Трансляция исходного программного кода в код на промежуточном языке программирования

Задачу трансляции исходного программного кода условно можно разделить на две подзадачи: трансляция одного файла и трансляция целого проекта. В первом случае достаточно получить пути, указанные пользователем в настройках, и запустить процесс компиляции с требуемыми ключами.

В свою очередь, трансляция проекта занимает продолжительное время, так как он может состоять из большого количества файлов. Вследствие этого работа распараллеливается согласно формуле $N - 1$, где N – количество ядер центрального процессора (ЦП). Это требуется для того, чтобы избежать полной загрузки ЦП для возможности работы других программ. Принцип трансляции всего проекта показан на рис. 1.

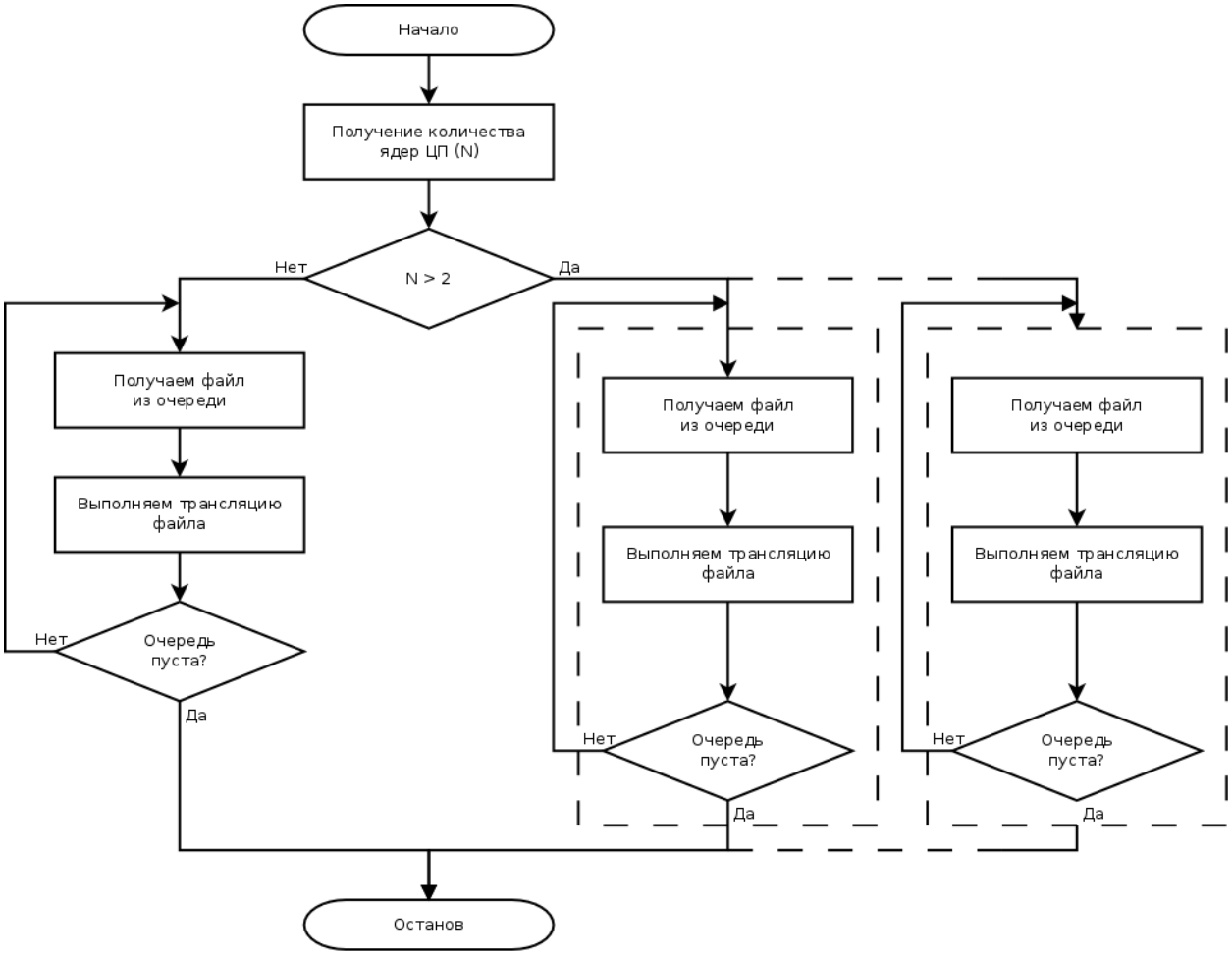

Рис. 1. Блок-схема алгоритма трансляции программного проекта

2.3. Снятие метрик кода программного обеспечения

Аналогично задаче трансляции задачу снятия метрик программного кода можно разделить на подзадачи анализа одной функции и нескольких.

При анализе отдельной функции мы извлекаем следующую информацию: количество обращений к регистровой (РП) и оперативной памяти (ОП), цикломатическая сложность, занимаемая память данных (ПД), количество вызовов внешних функций и асимптотическая сложность. Для дальнейшей нормализации значений метрик извлекается информация о количестве инструкций ЦП каждой функции.

Для нахождения первых двух метрик выполняется анализ каждой инструкции на наличие обращений к РП и/или ОП. При необходимости соответствующий счетчик инкрементируется.

Для нахождения цикломатической сложности используется формула $E - N + 2P$ [8], в которой за узлы ГПУ принимаются базовые блоки промежуточного представления функции, а за ребра – переходы между ними.

Количество требуемой ПД определяется путем анализа инструкций создания новой переменной и выделения памяти. Инструкции обоих типов содержат информацию об объеме выделяемой памяти в битах.

Подсчет количества вызовов функций выполняется путем инкрементации соответствующего счетчика при встрече соответствующей инструкции.

Определение асимптотической сложности осуществляется путем построения ГПУ функции (раздел 2.4), выделения в нем колец, среди которых отсеиваются вложенные последовательности, и анализа выходных условий циклов.

Анализ множества функций, как и в случае с задачей трансляции исходного кода, следует распараллеливать. Оптимальным решением является объединение работ по трансляции и анализу программного кода, когда потоки трансляции формируют очередь файлов, а потоки подсчета метрик ее обрабатывают. В этом случае алгоритм работы идентичен представленному на рис. 1.

2.4. Отображение графа потока управления

Построение ГПУ выбранной функции выполняется путем выявления зависимостей между базовыми блоками этой функции. Для этого мы последовательно просматриваем все базовые блоки и определяем связи, через которые может быть выполнен переход на данный блок. Таким образом, базовые блоки функции выступают в роли узлов ГПУ, а связи между ними – в роли его ребер.

3. Результаты

На основе перечисленных в разделах 1 и 2 положений нами был разработан программный продукт (рис. 2), выполняющий статический анализ программного кода на ЯП C. Продукт лицензирован согласно GNU GPL v3 [9] и доступен для свободного использования [10].

В качестве примера работы программы можно привести результаты анализа функции сортировки вставками и двух функций ядра *Linux* (таблица).

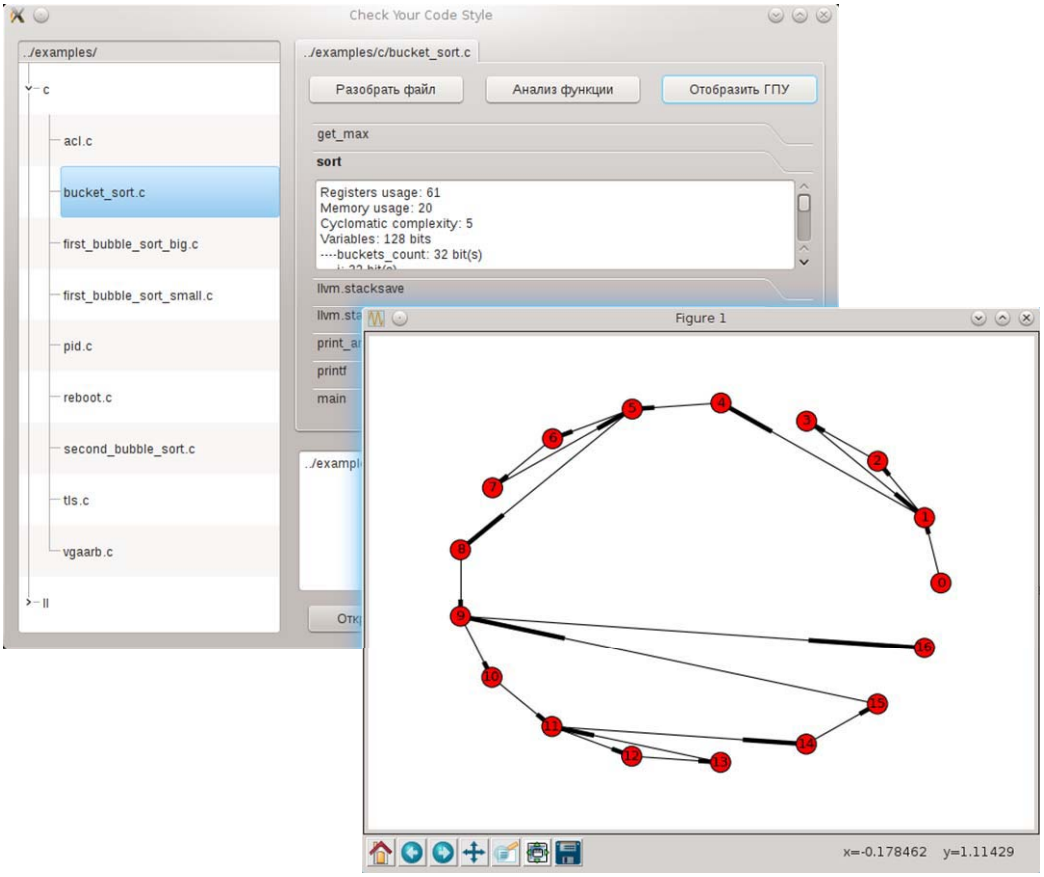

Рис. 2. Пример работы программного продукта

Результаты анализа некоторых функций

	Сортировка вставками	next_pidmap	__change_pid
Количество инструкций ЦП	84	48	42
Обращений к регистровой памяти	61	31	25
Обращений к оперативной памяти	20	12	16
Цикломатическая сложность	5	6	4
Требуемая память данных	128 бит	0 бит	0 бит
Вызовы функций	3	3	3

По полученным данным пользователь может сделать выводы о скорости выполнения анализируемых алгоритмов (больше инструкций ЦП, обращений к РП и ОП и вызовов внешних функций – более медленное выполнение программы), требуемой ОП (больше инструкций ЦП, данных и вызовов внешних функций – больший объем ОП требуется) и общей сложности выполнения программы (прямо пропорционально значению цикломатической и асимптотической сложностей), а также сравнивать алгоритмы между собой путем нормирования полученных метрик по объему данных и количеству инструкций ЦП.

Заключение

В данной статье было рассмотрено решение поставленной задачи о проектировании программного продукта для выполнения статического анализа программного кода на ЯП *C*. Были определены подзадачи и описаны принципы их решения. В качестве результата применения данных положений был представлен соответствующий программный продукт, готовый к применению в промышленной среде для получения количественных показателей качества разрабатываемого программного обеспечения.

Дальнейшая работа направлена на улучшение алгоритма определения асимптотической сложности программного кода и построение математической модели комплексной метрики программного кода.

Библиографические ссылки

1. *Вытовтов П. К.*, *Марков Е. М.* Программа оценки качества программного кода // Сб. тезисов докл. XVI Республ. выставки-сессии студенческих инновационных проектов (Ижевск, 12 ноября 2013 г.). – Ижевск, 2013. – С. 70–71.

2. *Вытовтов П. К.*, *Марков Е. М.* Разработка метода оценки программного кода и программного продукта, основанного на данном методе // Сб. тезисов докл. XVIII Республ. выставки-сессии студенческих инновационных проектов (Ижевск, 14 ноября 2014 г.). – Ижевск : ИННОВА, 2014. – 56 с.

3. *Вытовтов П. К.*, *Марков Е. М.* Сравнительный анализ методов оценки кода программного обеспечения // Информационные технологии в науке, промышленности и образовании : сб. тр. регион. науч.-техн. очно-заоч. конф. (Ижевск, 24 мая 2014 г.). – Ижевск : Изд-во ИжГТУ имени М. Т. Калашникова, 2014. – 364 с.

4. *Markov E. M.*, *M. Aiman Al Akkad*, *Vytovtov P. K.* Analysis of Software Code Metrics for Defining Their Priority for Cocol's Metric.

5. *Вытовтов П. К.* Разработка метода оценки вычислительной сложности и ресурсоемкости программного кода // Сб. тезисов докл. XVII Республ. выставки-сессии студенческих инновационных проктов (Ижевск, 10 апреля 2014 г.). – Ижевск, 2014. – С. 16–17.

6. *Vytovtov P. K.*, *Markov E. M.* Abstract syntax tree analysis software development for evaluating source code quality // Fourth Forum of Young Researchers. In the framework of International Forum "Education Quality – 2014". – Izhevsk : Publishing House of Kalashnikov ISTU, 2014. – Pp. 82–84.

7. LLVM Language Reference Manual – LLVM 3.7 Documentation. – URL: http://llvm.org/docs/LangRef.html (дата обращения: 19.04.15).

8. *Thomas J. McCabe.* A Complexity Measure // IEEE Transactions on Software Engineering. – Vol. SE-2. – No. 4. – December 1976. – Pp. 308–320.

9. The GNU General Public License v3.0 – GNU Project – Free Software Foundation. – URL: https://gnu.org/licenses/gpl.html (дата обращения: 19.04.15).

10. osanwe/check-your-code-style. – URL: https://github.com/osanwe/check-your-code-style (дата обращения: 19.04.15).

Получено 05.05.2015

УДК 614.254.7

Е. В. Дюжева, филиал (г. Ижевск) ФКУ НИИ ФСИН России
С. Б. Пономарев, доктор медицинских наук, ИжГТУ имени М. Т. Калашникова
К. А. Романов, филиал (г. Ижевск) ФКУ НИИ ФСИН России
Е. А. Белякова, студентка, ИжГТУ имени М. Т. Калашникова
И. А. Латыпова, магистрант, ИжГТУ имени М. Т. Калашникова

ОРГАНИЗАЦИОННО-УПРАВЛЕНЧЕСКИЙ АСПЕКТ МЕДИКО-САНИТАРНОГО ОБЕСПЕЧЕНИЯ ЛИЦ С АРТЕРИАЛЬНОЙ ГИПЕРТЕНЗИЕЙ, СОДЕРЖАЩИХСЯ В УЧРЕЖДЕНИЯХ УГОЛОВНО-ИСПОЛНИТЕЛЬНОЙ СИСТЕМЫ

За последние годы почти у 42 млн человек в нашей стране отмечен повышенный уровень артериального давления (АД), из них 39,9 % – среди мужчин и 41,1 % – среди женщин [1]. Более того, артериальная гипертензия (АГ) занимает первое место по вкладу в смертность от сердечно-сосудистых заболеваний (ССЗ) [2]. Чаще всего больные умирают от ее осложнений. Взаимосвязь между уровнем АД и риском ССЗ непрерывна, постоянна и не зависит от других факторов риска. Иными словами, чем выше АД, тем выше риск развития сердечно-сосудистых осложнений [3].

Кроме того, с середины прошлого века растет смертность от сердечно-сосудистых заболеваний, которые в структуре общей смертности населения России занимают 56,4 % [4]. Больше половины слу-