Вісник Національного технічного університету "ХПІ", 2018, № 24 (1300)
ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

УДК 004.415.52 **DOI: 10.20998/2411-0558.2018.24.04**

А. Ю. ЗАКОВОРОТНЫЙ, д-р техн. наук, проф., ученый секретарь НТУ "ХПИ",
М. А. ЗОРЯН, асп., НТУ "ХПИ"

МОДЕЛЬ НАКОПЛЕНИЯ УЯЗВИМОСТЕЙ В ИСХОДНОМ КОДЕ JAVA-ПРИЛОЖЕНИЙ, А ТАКЖЕ БИНАРНАЯ СИСТЕМА ОЦЕНКИ КАЧЕСТВА КОДА НА ЕЕ ОСНОВЕ

Статья посвящена проблеме оценки качества исходного кода Java-приложений и библиотек. В работе показано, что исследованный набор из 60956 GitHub репозиториев не подчиняется закону нормального распределения по количеству строк исходного кода. Разработанная модель накопления уязвимостей в исходном коде позволила создать бинарную систему оценки качества исходного кода для приложений и библиотек, написанных на Java. Ил.: 3. Табл.: 1. Библиогр.: 11 назв.

Ключевые слова: качество кода; уязвимость; модель накопления уязвимостей; Java; система оценки качества.

Актуальность и новизна работы. В настоящее время процесс разработки программного обеспечения на языке программирования Java, как и любой другой производственный процесс, нуждается в системе прогнозирования качества продукта, а также в системе оценки указанного качества в форме, удобной для понимания даже непрофильному в сфере разработки программного обеспечения специалисту.

В данной работе впервые представлена модель накопления уязвимостей в исходном коде Java-приложений и библиотек, а также построена бинарная система оценки качества кода на ее основе.

Постановка проблемы. На сегодняшний день практически не существует сфер деятельности, в которых бы не были непосредственно или опосредовано задействованы программные продукты. Программные продукты, как и любые другие произведения человеческой деятельности, имеют дефекты. В большинстве сфер человеческой производственной деятельности существуют системы контроля и оценки качества готовой продукции, а также полупродуктов. Программный код приложений и библиотек также является продуктом интеллектуального производства и нуждается в надлежащей системе контроля качества на всех этапах разработки приложения.

На сегодняшний день существуют системы анализа исходного кода, как, например, SonarQube, позволяющие собирать разнообразные метрики исходного кода, такие как, например, количество строк исходного кода, количество блокирующих, мажорных, минорных и

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

других уязвимостей, количество багов. Тем не менее, подобная система не является удобной в использовании для конечного потребителя готового Java-приложения или библиотеки. Поскольку конечный потребитель, который также может выступать в роли заказчика разработки приложения, далеко не всегда является профильным специалистом в области разработки программного обеспечения, то он нуждается в системе маркировки качества исходного кода, а также в системе оценки затрат инвестиций на разработку вновь создаваемого программного продукта с учетом затрат на устранение возникающих в исходном коде багов и уязвимостей, а также затрат на поддержание и масштабирование готового программного продукта.

Анализ литературы. На сегодняшний день существует множество работ, посвященных исследованию исходного кода приложений, написанных на разных языках программирования [1 – 5]. Основная задача указанных работ заключается, например, в сравнительном анализе применимости языков программирования для решения специализированных задач физики и математики [3] с позиции удобства использования, выразительной силы и, как следствие, минимизации количества допускаемых ошибок в процессе разработки программного продукта.

Значительная часть работ [5] посвящена сбору общих сведений на платформе GitHub о применении языков программирования, таких как количество строк кода, количество проектов, в которых участвует разработчик, скорость исправления багов по запросам и т.д. Из работы [5] следует, что основным источником обнаружения и сигнализирования о возникающих в коде уязвимостях и багах является сообщество пользователей этого кода. Это и не удивительно, поскольку GitHub является крупнейшей на сегодняшний день платформой для размещения исходного кода, как проприетарного, так и открытого [5].

Качество кода в наиболее широком смысле этого термина интересует научное сообщество в не меньшей мере, чем сравнение языков программирования по степени подверженности появлению уязвимостей в исходном коде [6 – 10].

Так, работа [9] посвящена исследованию качества кода на платформе GitHub для различных языков программирования. Методика определения количества уязвимостей и багов, использованная в работе [9], основывается на выявлении ключевых слов, ассоциированных с наличием исправляемой уязвимости, в журнале коммитов. По нашему мнению, подобный подход не дает объективного представления о распространении уязвимостей в исходном коде, поскольку позволяет иметь данные об уязвимостях только постфактум, что не позволит

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

построить предиктивную модель оценки качества кода. Подобные же методики оценки качества кода используются в работах [6, 7].

Таким образом, возникает проблема выбора подхода для априорной оценки наличия в исходном коде уязвимостей, то есть такого подхода, который укажет на наличие бага или уязвимости еще до того, как этот баг будет обнаружен разработчиками, тестировщиками или, что еще хуже, конечными пользователями программного продукта.

Также обзор как отечественных, так и зарубежных источников, указывает на отсутствие системы оценки качества кода, доступной для использования непрофильными специалистами.

Целью настоящей работы является разработка модели накопления уязвимостей в исходном коде Java-приложений и библиотек с последующим построением бинарной системы оценки качества кода на ее основе.

Источники данных и инструменты. Наиболее популярными платформами для размещения исходного кода являются GitHub, Bitbucket и SourceForge [8]. Для исследования качества кода приложений и библиотек, написанных на Java, нами был выбран именно GitHub, поскольку GitHub имеет хорошее API и огромный объем приложений и библиотек в открытом доступе.

Для сбора статистических данных о качестве кода нами был использован SonarQube, который является, по заявлению авторов, платформой управления качеством с открытым исходным кодом, нацеленным на непрерывный анализ и измерение технического качества исходного кода.

С помощью SonarQube для целей исследования нами были просканированы 60956 репозиториев, содержащих исходный код на Java, общим количеством строк кода 276 015 051. В процессе сканирования нами были измерены такие метрики, как блокирующие уязвимости (V_B), критические уязвимости (V_C), мажорные уязвимости (V_M) и минорные уязвимости (V_m). Исходя из полученных данных нами были составлены такие оценки качества кода, как общая уязвимость (V_T) и удельная уязвимость (V_S):

$$V_T = V_B + V_C + V_M + V_m, \tag{1}$$

$$V_S = \frac{V_T}{SLOC}, \tag{2}$$

где *SLOC* – количество строк исходного кода.

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

Визуализация полученных данных. Визуализация распределения сканированных репозиториев с исходным кодом, написанным на Java, представлена на рис. 1.

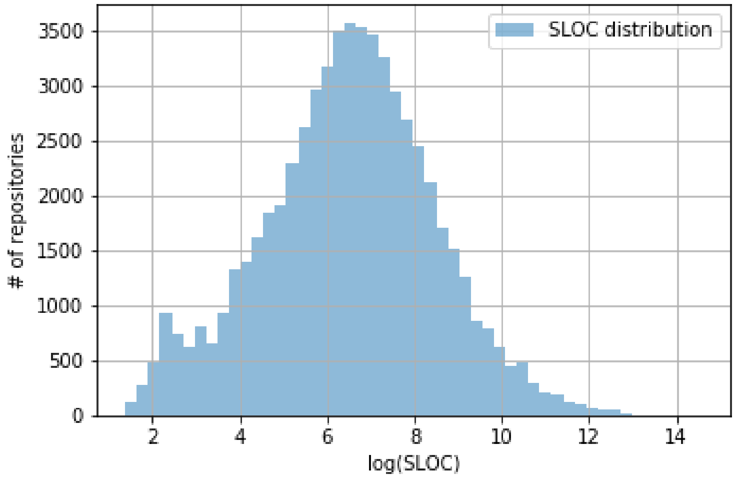

Рис. 1. Распределение просканированных репозиториев по количеству строк исходного кода

Приведённые на рис. 1 данные свидетельствуют, что 50% всех репозиториев имеют количество строк кода от $Q1$ = 5.26 ($SLOC_{Q1}$ = 192) до $Q3$ = 7.79 ($SLOC_{Q3}$ = 2416), медианное значение составляет $SLOC_m$ = 713. Также из рис. 1 очевидно, что распределение просканированных репозиториев по количеству строк кода не является нормальным.

Ключевыми факторами, определяющим вероятность появления уязвимости или бага являются общая структурная и архитектурная сложность проекта, выбранный язык программирования (компилируемый, интерпретируемый), выбранные фреймворк и стек зависимостей и сторонних библиотек. Очевидно, что структурная сложность проекта увеличивается с ростом числа строк кода *SLOC*. В текущем исследовании именно количество строк кода *SLOC* является основным аргументом, определяющим вероятность появления уязвимости и багов.

Рассмотрим графическое представление зависимости $V_T = f(SLOC)$ и $V_S = f(SLOC)$. На рис. 2 приведены данные сканирования репозиториев приложений и библиотек, написанных на Java.

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

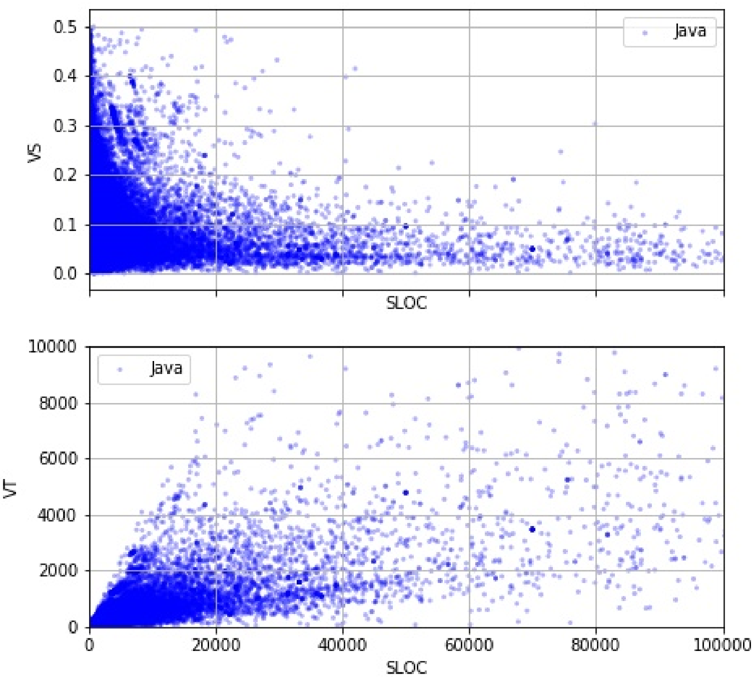

Рис. 2. Зависимости общей уязвимости V_T и удельной уязвимости V_S от количества строк исходного кода *SLOC*

Несмотря на то, что в случае $V_S = f(SLOC)$ наблюдается явная гиперболическая зависимость, мы не можем с уверенностью говорить, что чем больше объем репозитория, тем меньше общее количество уязвимостей, поскольку в данном случае имеет место влияние скрытых переменных [11]. Действительно, если мы рассмотрим уравнение (2), то будет очевидно, что в само определение V_S заложена гиперболическая зависимость от *SLOC*.

Анализ результатов сканирования. Очевидно (рис. 2), что увеличение количества строк кода в репозитории приводит к накоплению количества уязвимостей. Построение модели, описывающей характер накопления уязвимостей, может быть сведено в нашем случае к построению уравнения регрессии, которое опишет линию, проходящую через область с наибольшей плотностью вероятности появления в исходном коде уязвимости.

Зависимость $V_T = f(SLOC)$ может быть линеаризована следующим образом (рис. 3).

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

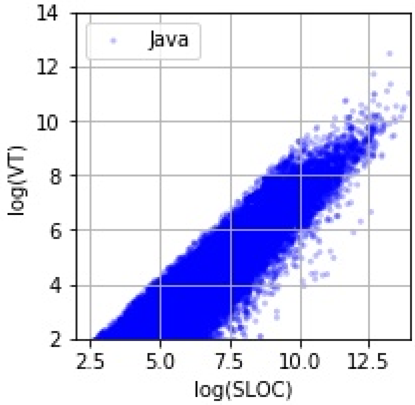

Рис. 3. Линеаризация $V_T = f(SLOC)$

Как видно из рис. 3 данные о накоплении уязвимостей в исходном коде Java-приложений и библиотек хорошо линеаризируются в логарифмических координатах и могут быть выражены следующим уравнением:

$$\log(V_T) = c + k\log(SLOC), \tag{3}$$

где c и k – эмпирические константы.

Результаты регрессионного анализа данных на рис. 3 приведены в табл. 1.

Таблица 1

Регрессионный анализ накопления уязвимостей

Коэффициент	Значение коэффициента	Стандартная ошибка	t-критерий	Доверительный интервал	
c	–1.9681	0.011	–181.019	–1.989	–1.947
k	0.9127	0.002	569.906	0.910	0.916

Статистические параметры уравнения (3), такие как R^2 = 0.842, $R_{adj}{}^2$ = 0.842, F = 3.248e+05, p = 0.000 (как для c, так и для k), указывают на то, что уравнения (3) адекватно описывает зависимость количества накопленных уязвимостей от количества строк кода репозитория.

Бинарная система оценки качества кода. Полученное уравнение (3) позволяет создать бинарную шкалу оценки (маркировки) качества

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

кода (**A**/**B**-классы), в основе которой лежит сравнение со средним количеством уязвимостей, характерных для указанного количества строк исходного кода Java-приложения. Качество кода в классе **A** выше, чем в классе **B**.

Так, например, если для репозитория известно количество строк кода l и количество уязвимостей v, то в случае $\log(v)$ c + k $\log(l)$, то исходному коду может быть присвоен класс A.

Выводы. Анализ открытых данных GitHub-репозиториев по языку программирования Java указывает на невозможность описать распределение репозиториев по количеству строк кода с помощью нормального распределения. Собранные статистические данные по распределению уязвимостей в Java приложениях позволили разработать модель накопления уязвимостей в исходном коде. Созданная в работе модель накопления уязвимостей послужила основой для разработки бинарной системы оценки качества кода. Бинарная система оценки качества кода позволяет присвоить исходному коду, написанному на Java, класс качества и, следовательно, ранжировать как готовые приложения, так и приложения, находящиеся в активной фазе разработки.

Использование разработанной нами бинарной системы оценки качества кода является удобным даже для неспециалиста в области разработки программного обеспечения.

Список литературы:

1. *Prechelt L.* An empirical comparison of seven programming languages / *L. Prechelt* // Computer. – 2000. – Т. 33. – №. 10. – С. 23-29.

2. *Nanz S.* A comparative study of programming languages in Rosetta Code / *S. Nanz, C.A. Furia* // Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on. – IEEE, 2015. – Т. 1. – С. 778-788.

3. *Sherin B.L.* A comparison of programming languages and algebraic notation as expressive languages for physics / *B.L. Sherin* // International Journal of Computers for Mathematical Learning. – 2001. – Т. 6. – №. 1. – С. 1-61.

4. *Prechelt L.* An empirical comparison of C, C++, java, perl, python, rexx and tcl / *L. Prechelt* // IEEE Computer. – 2000. – Т. 33. – № 10. – С. 23-29.

5. *Geiger R.S.* Summary analysis of the 2017 github open source survey / *R.S. Geiger* // arXiv preprint arXiv:1706.02777. – 2017.

6. *Gyimesi P.* Characterization of source code defects by data mining conducted on GitHub / *P. Gyimesi* // International Conference on Computational Science and Its Applications. – Springer, Cham, 2015. – С. 47-62.

7. *Kapur R.* Estimating defectiveness of source code: A predictive model using GitHub content, 2018 / *R. Kapur, B. Sodhi* // arXiv preprint arXiv:1803.07764.

8. *Counsell S.* Assert Use and Defectiveness in Industrial Code / *S. Counsell* // 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). – IEEE, 2017. – С. 20-23.

9. *Ray B.* A large scale study of programming languages and code quality in github / *B. Ray* // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. – ACM, 2014. – C. 155-165.
10. *Kalliamvakou E.* The promises and perils of mining github / *E. Kalliamvakou* // Proceedings of the 11th working conference on mining software repositories. – ACM, 2014. – C. 92-101.
11. *Pearl J.* Simpson's paradox, confounding, and collapibility / *J. Pearl* // Causality: models, reasoning and inference. – 2000. – № 7. –C. 173-200.

Bibliography:
1. Prechelt, L. (2000), "An empirical comparison of seven programming languages", *Computer*, Vol. 33, No. 10, pp. 23-29.
2. Nanz, S., and Furia, C.A. (2015), "A comparative study of programming languages in Rosetta Code", *Software Engineering (ICSE), IEEE/ACM 37th IEEE International Conference, IEEE*, Vol. 1, pp. 778-788.
3. Sherin, B.L. (2001), "A comparison of programming languages and algebraic notation as expressive languages for physics", *International Journal of Computers for Mathematical Learning.* Vol. 6, No. 1, pp. 1-61.
4. Prechelt, L. (2000), "An empirical comparison of c, C++, Java, Perl, Python, Rexx and Tcl", Vol. 33, No. 10, pp. 23-29.
5. Geiger, R.S. (2017), Summary analysis of the 2017 github open source survey, Arxiv preprint arXiv:1706.02777.
6. Gyimesi, P. (2015), "Characterization of source code defects by data mining conducted on GitHub", *International Conference on Computational Science and Its Applications*, Springer, Cham, pp. 47-62.
7. Kapur, R., and Sodhi, B. (2018), Estimating defectiveness of source code: A predictive model using GitHub content, Arxiv preprint arXiv:1803.07764.
8. Counsell, S. (2017), "Assert Use and Defectiveness in Industrial Code", *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* IEEE, pp. 20-23.
9. Ray, B. (2014), "A large scale study of programming languages and code quality in github", *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM, pp. 155-165.
10. Kalliamvakou, E. (2014), "The promises and perils of mining github", *Proceedings of the 11th working conference on mining software repositories*, ACM, pp. 92-101.
11. Pearl, J. (2000), "Simpson's paradox, confounding, and collapibility", *Causality: models, reasoning and inference*, No. 7, pp. 173-200.

Статью представил д-р техн. наук, проф. НТУ "ХПИ" Леонов С.Ю.

Поступила (received) 19.05.2018

Zakovorotnyi Alexandr, Dr. Tech. Sci., Sci. Secretary
National Technical University “Kharkov Polytechnic Institute”
Str. Kirpichova, 2, Kharkov, Ukraine, 61002
Tel.: +38 (097) 967-32-71, e-mail: zakovorotniy@kpi.kharkov.ua
ORCID ID: 0000-0003-4415-838x

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

Zorian Max, postgraduate
National Technical University “Kharkov Polytechnic Institute”
Str. Kirpichova, 2, Kharkov, Ukraine, 61002
Tel.: +38 (097) 560-62-00, e-mail: zorianmax@gmail.com
ORCID ID: 0000-0001-9676-3843

УДК 004.415.52

Модель накопичення вразливостей у вихідному коді JAVA-додатків, а також бінарна система оцінки якості кода на її основі / О.Ю. Заковоротний, **М.О. Зорян** // Вісник НТУ "ХПІ". Серія: Інформатика та моделювання. – Харків: НТУ "ХПІ". - 2018. – № 24 (1300). - С. 37 – 46.

Статтю присвячено проблемі оцінки якості вихідного коду Java-додатків і бібліотек. В роботі показано, що досліджений набір з 60956 GitHub репозиторіїв не описується законом нормального розподілу за кількістю рядків вихідного коду. Розроблена модель накопичення вразливостей у вихідному коді від кількості рядків вихідного коду дозволила створити бінарну систему оцінки якості початково коду для додатків і бібліотек, написаних на Java. Іл.: 3. Табл.: 1. Бібліогр.: 11 назв.

Ключові слова: якість коду, вразливість, модель накопичення уразливості, Java, система оцінки якості.

УДК 004.415.52

Модель накопления уязвимостей в исходном коде JAVA-приложений, а также бинарная система оценки качества кода на ее основе / А.Ю. Заковоротный, **М.А. Зорян** // Вестник НТУ "ХПИ". Серия: Информатика и моделирование. – Харьков: НТУ "ХПИ". – 2018. – № 24 (1300). – С. 37 – 46.

Статья посвящена проблеме оценки качества исходного кода Java-приложений и библиотек. В работе показано, что исследованный набор из 60956 GitHub репозиториев не подчиняется закону нормального распределения по количеству строк исходного кода. Разработанная модель накопления уязвимостей в исходном коде от количества строк исходного кода позволила создать бинарную систему оценки качества исходно кода для приложений и библиотек, написанных на Java. Ил.: 3. Табл.: 1. Библиогр.: 11 назв.

Ключевые слова: качество кода; уязвимость; модель накопления уязвимостей; Java; система оценки качества.

UDC 004.415.52

Model of vulnerability accumulation in source code of JAVA applications and binary system of estimation of the quality of the source code based on it / A.Yu. Zakovorotnyi, **M.A. Zorian** // Herald of the National Technical University "KhPI". – 2018. – № 24 (1300). – P. 37 – 46.

The article is devoted to the problem of evaluation of the quality of the source code of Java applications and libraries. The article shows that the investigated set of 60,956 GitHub repositories does not fit the normal distribution law by the number of lines of source code. The developed model of accumulation of vulnerabilities in the source code from the number of lines of source code allowed creating a binary system for assessing the quality of source code for Java-based applications and libraries. Figs.: 3. Tabl.: 1. Refs.: 11 titles.

Keywords: code quality; vulnerability; vulnerability model; Java; quality assessment system.