www.volsu.ru

DOI: https://doi.org/10.15688/NBIT.jvolsu.2024.4.4

УДК 004.056
ББК 67.401

СБОР МЕТРИК ПРОГРАММНОГО КОДА ДЛЯ АНАЛИЗА ЕГО УЯЗВИМОСТЕЙ

Глеб Александрович Попов

Старший преподаватель, кафедра информационной безопасности,
Волгоградский государственный университет
gpopov@volsu.ru
просп. Университетский, 100, 400062 г. Волгоград, Российская Федерация

Мария Михайловна Жунёва

Студент, кафедра информационной безопасности
Волгоградский государственный университет
bit-201_644968@volsu.ru
просп. Университетский, 100, 400062 г. Волгоград, Российская Федерация

Аннотация. Сбор метрической информации о коде является одним из наиболее доступных методов статического анализа, позволяющим выявлять потенциальные ошибки и уязвимости в программном обеспечении. Основные метрики, используемые в этом процессе, включают количество строк комментариев, иерархию наследования, цикломатическую сложность и вычислительную сложность. Количество комментариев помогает оценить понятность кода, в то время как сложные схемы наследования могут привести к трудностям в поддержке и увеличению вероятности ошибок. Цикломатическая сложность, предложенная Томасом Мак-Кейбом, измеряет количество независимых путей выполнения в коде, что позволяет оценить его сложность и потенциальные риски. Метрики Холстеда, основанные на статистическом анализе операторов и операндов, помогают предсказать количество ошибок в программе. Важно отметить, что сбор метрической информации не гарантирует отсутствие ошибок, а лишь указывает на участки кода, требующие внимания. Процесс сбора метрик прост в реализации и не требует значительных усилий со стороны разработчиков, что делает его ценным инструментом для повышения качества программного обеспечения. В заключение, использование метрических данных в анализе кода способствует более эффективному выявлению и устранению потенциальных проблем в программных системах.

Ключевые слова: метрическая информация, статический анализ кода, ошибки программного обеспечения, метрики Холстеда, анализ кода.

Сбор метрической информации о коде – один из самых легко реализуемых методов статического анализа кода. На основе собранных данных делается вывод о среднем количестве ошибок и о месте их наиболее ожидаемого появления, а также определяются участки кода, на которые должно обращать особенное внимание. Собираемая метрическая информация включает следующие метрики:

– количество строк комментариев (чаще всего – по отношению к строкам кода). Идея состоит в том, что код, содержащий малое количество комментариев, труден для понимания (особенно для разработчиков, которые сами не писали его) и, поэтому, может содержать незамеченные ошибки. Ко всему прочему, правильную работу такого кода можно легко нарушить неосторожным изменением какой-либо его части. С появлением автоматических инструментов генерации документации из исходного кода данная метрика стала еще более важной, так как теперь стало возможным вычислять ее для отдельных программных компонентов;

– иерархия наследования при использовании объектно-ориентированного подхода. Запутанная и сложная схема наследования классов также приводит к трудностям в понимании и поддержкой кода, что, в свою очередь, увеличивает вероятность нахождения ошибки в этих частях кода. Данная метрика вычисляет глубину наследования (количество классов-предков) и количество классов, имеющих общего родителя. Классы, имеющие длинное генеалогическое дерево аккумулируют в себя ошибки классов-предшественников, поэтому им должно уделяться особое внимание со стороны разработчиков. Аналогично, класс, имеющий множество наследников, должен быть также верифицирован тщательным образом. Для языков с поддержкой множественного наследования (например, C++) эта метрика играет особо важную роль, показывая сколько родительских классов имеет дочерний класс;

– цикломатическая сложность. Термин, как и сама метрика, была предложена Томасом Мак-Кейбом в 1976 году. Представляет собой количество линейно независимых путей выполнения на рассматриваемом участке программы. Анализируемый участок может быть отдельной функцией, классом или целым модулем. Цикломатическая сложность вычисляется на основе графа потока управления (CFG). Узлами CFG являются базовые блоки кода (участки кода, не содержащие ветвлений), а ребра – переходы управления между базовыми блоками. Значение цикломатической сложности рассчитывается по следующей формуле:

$$M = E - N + 2 \times P, \quad (1)$$

где M – значение циклической сложности; E – количество ребер в графе; N – количество узлов в графе; P – компонента связности графа. В своей работе «A Complexity Measure» Мак-Кейб рекомендует перерабатывать код приложения таким образом, чтобы цикломатическая сложность не превышала значения 10. С момента написания этой работы в 1976 г. эта практика пересматривалась многими специалистами, в том числе и Национальным институтом стандартов и технологий США, однако рекомендуемое значение до сих пор остается прежним;

– вычислительная сложность. Определение сложности фрагмента кода в виде О-нотации позволяет делать выводы о взаимозависимости входных параметров. Если формула сложности содержит произведение каких-либо входных параметров, это говорит об их взаимозависимости. В большинстве случаев, алгоритмы, имеющие большое количество взаимозависимостей во входных данных сложны для разработки и понимания и, как следствие, часто содержат ошибки. Если же формула сложности включает в себя множество слагаемых, то описываемый алгоритм содержит в себе независимые вычисления, которые необходимо разносить по отдельным программным модулям для упрощения их поддержки [1];

– метрики Холстеда. Метрическая теория программ М.Х. Холстеда исходит из статистического выражения алгоритма на конкретном языке программирования. В своей работе «Elements of software science» [2] он предлагает следующие характеристики исходных кодов: η_1 – число простейших операторов в программе; η_2 – число простейших операндов; $N1$ – общее число операторов; $N2$ – общее число операндов; f_{j1} – число вхождений j-го наиболее часто встречающегося оператора ($j = \overline{1,\eta_1}$); f_{j2} – число вхождений j-го наиболее часто встречающегося оператора ($j = \overline{1,\eta_2}$). Отправляясь от этих базовых метрических характеристик Холстед определяет так же понятия словаря

$$\eta = \eta_1 + \eta_2, \quad (2)$$

длины реализации алгоритма

$$N = N_1 + N_2, \quad (3)$$

и предлагает формулы, связывающие эти величины между собой и позволяющие вычислить их на практике. Определяя объем программы как

$$V = N \log_2 \eta, \quad (4)$$

Холстед отмечает, что этот параметр не может однозначно характеризовать реализацию алгоритма ввиду наличия избыточности в программном коде. На основании данного вывода он вводит понятия потенциальных (минимальных) метрических характеристик (η, V^* и т. д.), описывающих данный алгоритм в наиболее сжатой его форме. Отличие данной реализации алгоритма от потенциального Холстед выразил понятием уровня программы

$$L = \frac{V^*}{V} \quad (5)$$

и вывел формулу для его практического расчета

$$\hat{L} = \frac{\eta_{1^*}}{\eta_1} \frac{\eta_2}{N_2}. \quad (6)$$

Наконец, Холстед определяет важнейшую характеристику языка программирования – уровень языка

$$\Pi = LV^* = L^2 V. \quad (7)$$

Одним из приложений теории Холстеда является гипотеза ошибок [3], с помощью которой можно предсказать ожидаемое количество ошибок в программе. Автор предлагает следующую формулу для оценки количества переданных программе ошибок:

$$\hat{B} = \frac{E_{\text{крит}}(V^*)}{\lambda^3}, \quad (8)$$

где $V = (\eta_1 + \eta_2)\log_2(\eta_1 + \eta_2)$ – потенциальный объем программы, а $E_{крит}$ – число элементарных мысленных различений, требуемых для ее порождения.

Последняя характеристика отражает способность мозга удерживать в памяти несколько объектов одновременно. В заключении Холстед приводит расчеты для английского языка ($\lambda = 2{,}16$, $E_{крит} = 3000$), подтверждаемые экспериментальными исследованиями.

Подводя итог к вышеприведенному описанию метрик программного кода, можно выделить следующие существенные факты:

1. Метрики программного кода собираются на основе совокупности всех исходных файлов проекта, а потому обладают потенциальной возможностью 100%-ного покрытия кода.

2. Результатом обработки метрических данных является не отчет об найденных ошибках, а лишь указание на места их наиболее вероятного нахождения. Таким образом, анализ кода на наличие уязвимостей при помощи метрических данных не может обеспечить отсутствие ошибок первого и второго родов, а также не предоставляет возможности верификации результатов.

3. Средства сбора метрической информации достаточно просты в разработке. Наиболее сложная их часть, семантический анализатор, может быть, как разработан с нуля путем определения грамматики анализируемого языка, так и получен модификацией существующего языкового средства. В первом случае, он не обязан поддерживать весь синтаксис языка программирования, что значительно облегчает его реализацию.

4. Процесс сбора метрической информации не требует дополнительных усилий со стороны разработчиков.

В заключение, сбор метрической информации о коде представляет собой важный и эффективный инструмент для статического анализа программного обеспечения. Использование различных метрик, таких как количество строк комментариев, иерархия наследования, цикломатическая и вычислительная сложность, позволяет разработчикам выявлять потенциальные ошибки и уязвимости на ранних стадиях разработки. Эти метрики не только помогают оценить качество кода, но и указывают на участки, требующие особого внимания, что способствует более эффективному управлению процессом разработки и поддержкой программных систем.

Несмотря на то что сбор метрической информации не гарантирует полное отсутствие ошибок, он предоставляет ценную информацию для улучшения качества программного обеспечения и повышения его надежности. Простота реализации методов сбора метрик делает их доступными для широкого круга разработчи-

ков, что позволяет интегрировать их в существующие процессы разработки без значительных затрат времени и ресурсов. Таким образом, применение метрических данных в анализе кода является перспективным направлением, способствующим созданию более качественных и безопасных программных продуктов, что в конечном итоге повышает удовлетворенность пользователей и снижает риски, связанные с эксплуатацией программного обеспечения.

REFERENCES

1. Arora S., Barak B. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. 489 p. DOI: 10.1017/CBO9780511804090

2. Floyd R.W. *Assigning Meanings to Programs*. Dordrecht, Springer Netherlands, 1993, pp. 65-81. DOI: 10.1007/978-94-011-1793-7_4

3. Halstead M.H. *Elements of Software Science*. New York, North Holland, 1979. 127 p.

COLLECTING METRICS OF SOFTWARE CODE TO ANALYZE ITS VULNERABILITIES

Gleb A. Popov

Senior Lecturer, Department of Information Security,
Volgograd State University
gpopov@volsu.ru
Prosp. Universitetsky, 100, 400062 Volgograd, Russian Federation

Maria M. Zhuneva

Student, Department of Information Security,
Volgograd State University
bit-201_644968@volsu.ru
Prosp. Universitetsky, 100, 400062 Volgograd, Russian Federation

Abstract. Collecting metric information about the code is one of the most accessible methods of static analysis, which allows you to identify potential errors and vulnerabilities in the software. The main metrics used in this process include the number of comment lines, inheritance hierarchy, cyclomatic complexity, and computational complexity. The number of comments helps to assess the clarity of the code, while complex inheritance schemes can lead to difficulties in support and increase the likelihood of errors. Cyclomatic complexity, proposed by Thomas McCabe, measures the number of independent execution paths in the code, which allows you to assess its complexity and potential risks. Halsted metrics, based on statistical analysis of operators and operands, help predict the number of errors in a program. It is important to note that collecting metric information does not guarantee the absence of errors, but only indicates code sections that require attention. The process of collecting metrics is easy to implement and does not require significant efforts on the part of developers, which makes it a valuable tool for improving the quality of software. In conclusion, the use of metric data in code analysis contributes to more effective identification and elimination of potential problems in software systems.

Key words: metric information, static code analysis, software errors, Halsted metrics, code analysis.