УДК 681.326

В. В. Варениц а

ПРОБЛЕМЫ ВЫЧИСЛЕНИЯ МЕТРИК СЛОЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ ПРОВЕДЕНИИ АУДИТА БЕЗОПАСНОСТИ КОДА МЕТОДОМ РУЧНОГО РЕЦЕНЗИРОВАНИЯ

Представлен обзор метрик сложности программного обеспечения. Рассмотрена возможность использования метрик при оценке показателей качества программ.

E-mail: iitroickiy@mail.ru

Ключевые слова: сложность программ, метрики сложности

К настоящему времени разработано огромное количество средств автоматизации для проведения аудита кода программного обеспечения (ПО) в целях выявления программных закладок и уязвимостей в коде. Однако, несмотря на широкий выбор средств автоматизации, самым надежным и точным методом до сих пор является экспертный аудит кода (ручное рецензирование кода ПО). В данной работе рассматриваются основные подходы (метрики) по оценке трудозатрат аудита безопасности кода и проблемы, связанные с применением этих метрик.

Метрики сложности программного обеспечения. Метрики сложности ПО принято разделять на три основные группы:

1) метрики размера программ;

2) метрики сложности потока управления программ;

3) метрики сложности потока данных программ.

Метрики первой группы базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. К наиболее известным метрикам этой группы относят число операторов программы, количество строк исходного текста, набор метрик Холстеда. Метрики второй группы основаны на анализе управляющего графа программы. Представителем этой группы является метрика Маккейба. Метрики третьей группы базируются на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина.

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Такие метрики основываются на LOC-оценках. При косвенном измерении программного продукта и процесса его разработки вместо подсчета LOC-оценок при этом рассматривается не размер, а функциональность продукта.

В практике создания и аудита ПО наибольшее распространение получили размерно-ориентированные метрики. В организациях, занятых разработкой программной продукции, для каждого проекта принято регистрировать следующие показатели:

– общие трудозатраты (в человекомесяцах, человекочасах);

– объем программы (в тысячах строк исходного кода — LOC);

– стоимость разработки и аудита;

– объем документации;

– ошибки, обнаруженные в течение года эксплуатации;

– количество людей, участвующих в разработке и аудите;

– срок проведения аудита кода.

Перечисленные выше метрики не универсальны и спорны. Особенно это относится к такому показателю, как LOC, который существенно зависит от используемого языка программирования и уровня подготовки разработчика. Изначально данный показатель возник как способ оценки объема работы по проекту, в котором применялись языки программирования с достаточно простой структурой: «одна строка кода = одна команда языка». Также давно известно, что одну и ту же функциональность можно запрограммировать разным количеством строк. На языке высокого уровня (C++, Java) ту жу функциональность можно написать в пять-шесть строк. Однако современные средства программирования сами генерируют тысячи строк кода на простые операции. Потому метод LOC является только оценочным, а не обязательным.

Наиболее интересными являются *метрики стилистики и понятности программ*, так как важно не просто подсчитать количество строк комментариев в коде и просто соотнести с логическими строками кода, а узнать плотность комментариев, т. е. код сначала был документирован хорошо, затем — плохо. Или такой вариант: интерфейс функции или класса документирован и комментирован, а исходный код нет. Суть метрики проста: код разбивается на N-равные куски и для каждого из них определяется F_i:

$$F_i = \frac{N_i^{com}}{N_i - 0{,}1},$$

где N_i^{com} — количество закомментированных участков кода; N_i — общее количество проверяемых блоков кода.

Таким образом, общая оценка сложности кода F_{src} для проверяемой программы есть сумма F_i:

$$F_{src} = \Sigma F_i.$$

Основные проблемы, связанные с применением размерно-ориентированных метрик, которые актуальны на практике:

– оценку работы человека не всегда удается свести к нескольким числовым параметрам и по ним судить о производительности;

– метрики не учитывают опыт сотрудников и их личные качества;

– процесс измерения может быть искажен за счет того, что сотрудникам известны измеряемые показатели и они стремятся оптимизировать эти показатели, а не свою работу;

– нет метрик, которые были бы одновременно и значимы, и достаточно точны.

Количество строк кода — это просто количество строк, этот показатель не дает представления о сложности решаемой проблемы. Анализ функциональных точек был разработан в целях лучшего измерения сложности кода и спецификации, но он использует личные оценки измеряющего, поэтому разные люди получат различные результаты.

Для большинства программных проектов актуален объектно-ориентированный подход, в связи с чем существует значительное количество метрик, позволяющих получить оценку сложности объектно-ориентированных проектов. Наиболее актуальные из *объектно-ориентированных метрик* приведены в таблице.

Объектно-ориентированные метрики

Метрика	Описание
Взвешенная насыщенность класса 1 (Weighted Methods Per Class (WMC))	Отражает относительную меру сложности класса на основе цикломатической сложности каждого его метода. Класс с более сложными методами и большим количеством методов считается более сложным. При вычислении метрики родительские классы не учитываются
Взвешенная насыщенность класса 2 (Weighted Methods Per Class (WMC2))	Мера сложности класса, основанная на том, что класс с большим числом методов является более сложным и что метод с большим количеством параметров также является более сложным. При вычислении метрики родительские классы не учитываются
Глубина дерева наследования (Depth of Inheritance Tree)	Длина самого длинного пути наследования, заканчивающегося на данном модуле. Чем глубже дерево наследования модуля, тем сложнее предсказать его поведение. В то же время **увеличение глубины дает больший потенциал повторного использования данным модулем поведения, определенного для классов-предков**
Количество классов наследования (Number of Children)	Число модулей, непосредственно наследующих данный модуль. **Большие значения этой метрики указывают на широкие возможности по-**

Метрика	Описание
	вторного использования; при этом слишком большое значение может свидетельствовать о плохо выбранной абстракции
Связность объектов (Coupling between Objects)	Количество модулей, связанных с данным модулем в роли клиента или поставщика. Чрезмерная связность говорит о слабости модульной инкапсуляции и может препятствовать повторному использованию кода.
Отклик на класс (Response for Class)	Количество методов, которые могут вызываться экземплярами класса; вычисляется как сумма количества локальных методов или количества удаленных методов

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы. Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

1) NUOprtr (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

2) NUOprnd (Number of Unique Operands) — число уникальных операндов программы (словарь операндов);

3) Noprtr (Number of Operators) — общее число операторов в программе;

4) Noprnd (Number of Operands) — общее число операндов в программе.

На основании этих характеристик рассчитываются следующие оценки:

- словарь программы (Halstead Program Vocabulary, HPVoc): **HPVoc = NUOprtr + NUOprnd**;
- длина программы (Halstead Program Length, HPLen): **HPLen = = Noprtr + Noprnd**;
- объем программы (Halstead Program Volume, HPVol): **HPVol = = HPLen log2 HPVoc**;
- сложность программы (Halstead Difficulty, HDiff): **HDiff = (NUOprtr / 2) × (NOprnd / NUOprnd)**.

На базе показателя HDiff целесообразно оценивать трудозатраты эксперта, проводящего аудит кода HEff (Halstead Effort): **HEff = = HDiff × HPVol**.

Метрики цикломатической сложности по Мак-Кейбу позволяют не только произвести оценку трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показате-

ли оценки длительности и стоимости проекта, но и оценить связанные риски и принять необходимые управленческие решения. Упрощенная формула вычисления цикломатической сложности имеет вид:

$$C = E - N + 2,$$

где E — число ребер; N — число узлов на графе управляющей логики.

На практике при вычислении цикломатической сложности логические операторы не учитываются. В реальных проектах, как правило, применяется упрощенный подход, в соответствии с которым построение графа не осуществляется, а вычисление показателя проводится на основании подсчета числа операторов управляющей логики (if, switch и т. п.) и возможного количества путей исполнения программы. Цикломатическое число Мак-Кейба показывает требуемое количество проходов для покрытия всех контуров сильносвязанного графа или количество тестовых прогонов программы, необходимых для исчерпывающего тестирования по принципу «работает каждая ветвь».

Метрики Чепина позволяют оценить информационную прочность отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода. Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

1) Множество «*P*» — вводимые переменные для расчетов и обеспечения вывода;

2) Множество «*M*» — модифицируемые или создаваемые внутри программы переменные;

3) Множество «*C*» — переменные, участвующие в управлении работой программного модуля (управляющие переменные);

4) Множество «*T*» — не используемые в программе («паразитные») переменные.

Далее вводится значение метрики Чепина:

$$Q = a1P + a2M + a3C + a4T,$$

где $a1$, $a2$, $a3$, $a4$ — весовые коэффициенты.

С учетом весовых коэффициентов выражение примет вид

$$Q = P + 2M + 3C + 0{,}5T.$$

Для предварительной оценки сложности программы на этапе разработки (согласования) спецификации требований к программе может быть использована *метрика прогнозируемого числа операторов* N_{pr} программы:

$$N_{pr} = N_f \cdot N_1,$$

где N_f — количество функциональных требований к проверяемой программе; N_1 — статистическое единичное значение количества операторов (среднее число операторов, приходящихся на одно функциональное требование).

При проведении аудита безопасности кода также учитывается *метрика оценки сложности на этапе определения архитектуры* тестируемого ПО:

$$S = \frac{N_i}{N_f \cdot N_i^1 \cdot k_S},$$

где N_i — общее количество переменных, передаваемых по интерфейсу между компонентами ПО; N_i^1 — единичное значение количества передаваемых переменных; k_s — коэффициент сложности тестируемого ПО.

Заключение. Рассмотрены основные подходы (метрики) по оценке трудозатрат аудита безопасности кода и проблемы, связанные с их применением в реальных проектах. С одной стороны, многие из предложенных метрик не идеальны для использования на практике и имеют высокую погрешность, но с другой стороны, без использования метрик оценки сложности программного кода, который будет проверяться в целях выявления в нем уязвимостей и программных закладок, невозможно оценить сложность тестирования и спрогнозировать главный критерий — время на выполнение конкретного проекта.

СПИСОК ЛИТЕРАТУРЫ

1. The Build Master. Microsoft Software Configuration Management Best Practices.
2. Steven C. McConnell «Code Complete». – 2nd. ed.
3. IEEE Std 1045–1992. Standard for Software Productivity Metrics.
4. IEEE Std 1490–1998. Adoption of PMI Standard — A Guide to the Project Management Body of Knowledge.
5. Изосимов А. В., Рыжко А. Л. Метрическая оценка качества программ. – М.: МАИ, 1989.
6. Холстед М. Начала науки о программах. – М.: Финансы и статистика, 1981.
7. Марков А. С. Оценка динамической сложности программного обеспечения на ПЭВМ // Методы и средства совершенствования сложных управляющих систем и комплексов. – СПб.: БГТУ им. Д.Ф. Устинова (Военмех), 1992. –С. 14–27.
8. Марков А. С., Миронов С. В., Цирлов В. Л. Выявление уязвимостей в программном коде // Открытые системы. СУБД. – 2005. – № 12. – С. 64–69.
9. Марков А. С., Цирлов В. Л., Маслов В. Г., Олексенко И. А. Тестирование и испытания программного обеспечения по требованиям безопасности информации // Изв. Ин-та инж. физ. – 2009. – № 2 (12). – С. 2–6.

Статья поступила в редакцию 19.10.2011