АЛГОРИТМИЗЦИЯ И ПРОГРАММИРОВАНИЕ

УДК 681.391

В. А. Пласковицкий, аспирант (БГТУ)

ПРИМЕНЕНИЕ МЕТРИК ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ОЦЕНКИ СЛОЖНОСТИ ИСПОЛНЯЕМОГО КОДА

В статье рассмотрены практические реализации метрик программного обеспечения для оценки сложности исполняемого кода, что необходимо при выборе и разработке более эффективных методов защиты программного кода. Исследованы отдельные проблемы, связанные с реализацией сбора метрик, и их возможные решения. Рассмотрена работа метрик на примерах различных приложений. Актуальность работы связана с переходом хранения программных продуктов от скомпилированных в двоичный вид команд к хранению в промежуточном или исходном коде.

Practical realizations of software metrics for estimation of executable code complexity are given in the article, which is necessary for developing of efficient software protection methods. Single problems connected to realization of the metrics gathering and their possible solutions are investigated. Functioning of the metrics is given for different applications. Topicality of the research is provided by software storage passing from compiled binary form to portable or source code.

Введение. Современные программные продукты хранятся либо в виде промежуточного кода, исполняемого на виртуальных машинах (Java, C#, Ruby, ActionScript, Python), либо исходного кода, интерпретируемого на этапе выполнения (PHP, RoR, ASP.NET, JavaScript, HTML 5). Несмотря на все преимущества такого подхода, исполняемый код программных приложений становится значительно доступней для изучения, а значит, и для несанкционированного использования.

Шифрование, в том числе и с использованием технологии расшифрования «на лету» [1], не может предоставить защиты на этапе выполнения программы. Обфускация кода [2] действует на всех этапах работы приложения, но нет четких критериев оценки ее эффективности. К важнейшим из таких критериев относится оценка сложности анализа и изменения кода.

Метрики программного обеспечения не нашли широкого применения при оценке качества программного кода, но могут предоставить целый ряд показателей, оценивающих его сложность.

В работе рассматриваются проблемы, возникающие при использовании метрик программного обеспечения, подходы к их решению и существующие инструменты, реализующие их сбор.

Основная часть. Метрики программного обеспечения меры, позволяющие получить численное значение некоторого свойства программного обеспечения.

Поскольку при оценке сложности программного кода необходимо произвести его анализ, возникает проблема, связанная с синтаксисом языка, а также переносимостью разработок для одного языка на другой. К счастью, существует не так много действительно кардинально отличающихся синтаксисов, тем не менее, прямой перенос также маловероятен.

Безусловно, наилучшим решением являлось бы API от самих разработчиков языков программирования, но, к сожалению, чаще всего разработка подобного инструментария отдается на сторону разработчиков проприетарных дополнений. Вместо разработки собственных выражений для анализа синтаксиса языка можно воспользоваться уже готовыми инструментами, например, ANTLR. Кроме простоты использования, преимуществом такого подхода является уже готовые решения для целого ряда синтаксисов: Java, C#, Python, Ruby и др.

Существуют и другие синтаксические анализаторы, например, Source-Navigator и Source Insight, предоставляющие возможность описание синтаксиса C, C++, Java, Tcl.

Для оценки защиты программы с точки зрения простоты изменения кода, можно использовать анализ зависимостей и связностей объектов. Например:

– центростремительное сцепление (afferent coupling) – число типов или методов из внешних сборок, которые используют заданный тип или метод;

– центробежное сцепление (efferent coupling) – количество классов внутри этой категории, которые зависят от классов вне этой категории;

– нестабильность (instability) – отношение центробежного сцепления к сумме обоих сцеплений (чем ближе к 1, тем более нестабильна система);

– абстрактность категории – отношение количества абстрактных классов к их общему числу;

– расстояние от главной последовательности.

Более подробно с данными метриками можно ознакомиться в [4].

В пакет VisualStudio входит специальный инструмент Code Metrics, собирающий информацию о четырех видах метрик для приложений из результирующего IL-кода:

– цикломатическая сложность (Cyclomatic Complexity) – число линейно независимых маршрутов через программный код;

– глубина наследования (Depth of Inheritance) – число наследуемых классов;

– сцепление классов (Class Coupling) – зависимость классов друг от друга;

– количество строк кода (Lines of Code) – поскольку подсчет производится именно в IL-коде, то не учитываются строки с фигурными скобками, комментарии и т. п., зато сжатый синтаксис (например, LINQ-выражения) может развернутся на несколько строк.

Также в нем рассчитывается комплексный показатель качества кода MI (Maintainability Index), принимающий значения от 0 до 100.

Расчет происходит по формуле:

$$MI = MAX(0,(171 - 5,2 * \ln(HV) + -0,23 * CC - 16,2 * \ln(LoC)) * 100/171)), \quad (1)$$

где HV (Halstead Volume) – вычислительная сложность, пропорциональная числу операторов в коде; CC – цикломатическая сложность; LoC – количество строк кода.

Чем ниже показатель MI, тем сложнее производить поддержку кода (его анализ). Значения от 20 до 100 считаются удовлетворительными для понимания, от 10 до 19 – нежелательными, от 0 до 9 – тяжелыми для понимания. Пример использования метрик приведен на рис. 1. Схожий инструмент предоставляется и в свободно распространяемой среде Monodevelop.

Ряд интересных возможностей предлагает анализатор программных кодов «АК-ВС». Среди них: количество связей по управлению и информации, а также различные усредненные характеристики объектов. Так же в нем заявлена поддержка целого ряда языков программирования (C, C++, Java, C#). Но на данный момент он является слишком дорогостоящим продуктом с отсутствием пробной версии.

Инструмент VivaShowMetrics, работающий в рамках среды PVS-Studio для программ, разработанных на языке C++, предоставляет набор комплексных метрик для подсчета:

– общей сложности функций, базирующийся на основе ветвлений в теле функций, количестве вызовов других функций и количестве арифметических операций;

– сложности и опасности распараллеливания функции с использованием технологии OpenMP;

– сложности и опасности преобразования функции к 64-формату.

Пример работы приложения показан на рис. 2.

Вычисленные показатели метрик выводятся в виде трех соответствующих диаграмм сложности функции, распараллеливания и конвертации в 64-ричное представление.

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Prok (Debug)	75	282	7	103	1 526
{ } Prok	75	282	7	103	1 526
ControlWork	50	124	7	72	828
EmbedTools	68	4	1	3	18
GenKey	52	15	7	21	117
InfoChangeNode	100	1	1	0	1
InfoEmbedEncoding	93	6	1	1	9
InfoModule	91	6	1	2	11
Main	65	45	7	44	203
Node	89	6	1	0	8
Preloader	68	7	7	20	30
Program	81	1	1	3	3
RenameObject	63	6	7	15	36
RichTextBoxMark	92	3	1	3	4
RichTextBoxMark()	100	1		0	1
setMark(int, int) : void	84	1		3	2
setTF(RichTextBox) : void	95	1		1	1
View	64	58	7	24	258

Рис. 1. Сбор данных о метриках программного кода приложения «Prok» с помощью инструмента Code Metrics VisualStudio 2012

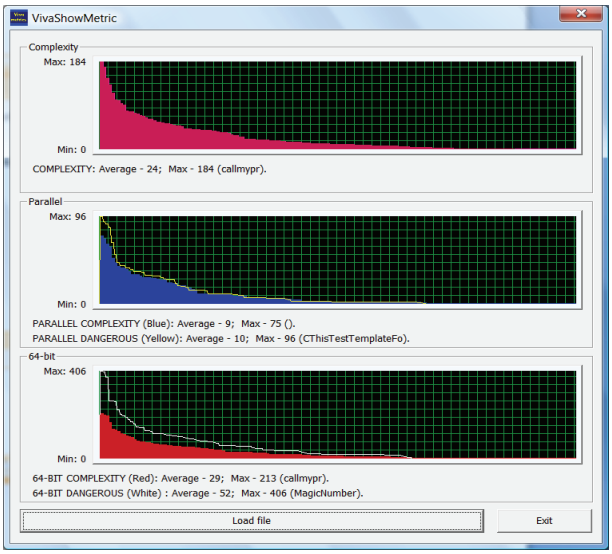


Рис. 2. Утилита VivaShowMetric
на примере анализа unit-тестов [3]

Инструмент IBM Rational ClearCase позволяет отобразить метрик на графе программного кода. При этом можно использовать произвольный инструмент, производящий сбор метрик с сохранением результатов в отдельный текстовый документ, например CCCC (рис. 3).

Сбор метрик может быть организован либо в режиме реального времени, когда вызов анализатора происходит в момент выполнения активного действия, например, регистрации изменений в репозитории, либо в отложенном режиме через запуск по расписанию.

Результаты могут храниться как в отдельной базе данных, так и непосредственно в ClearCase.

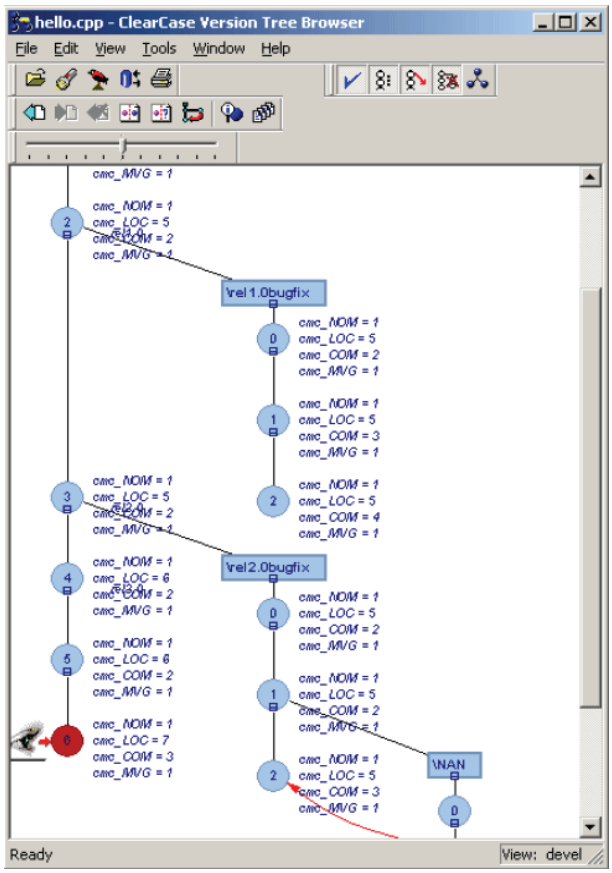
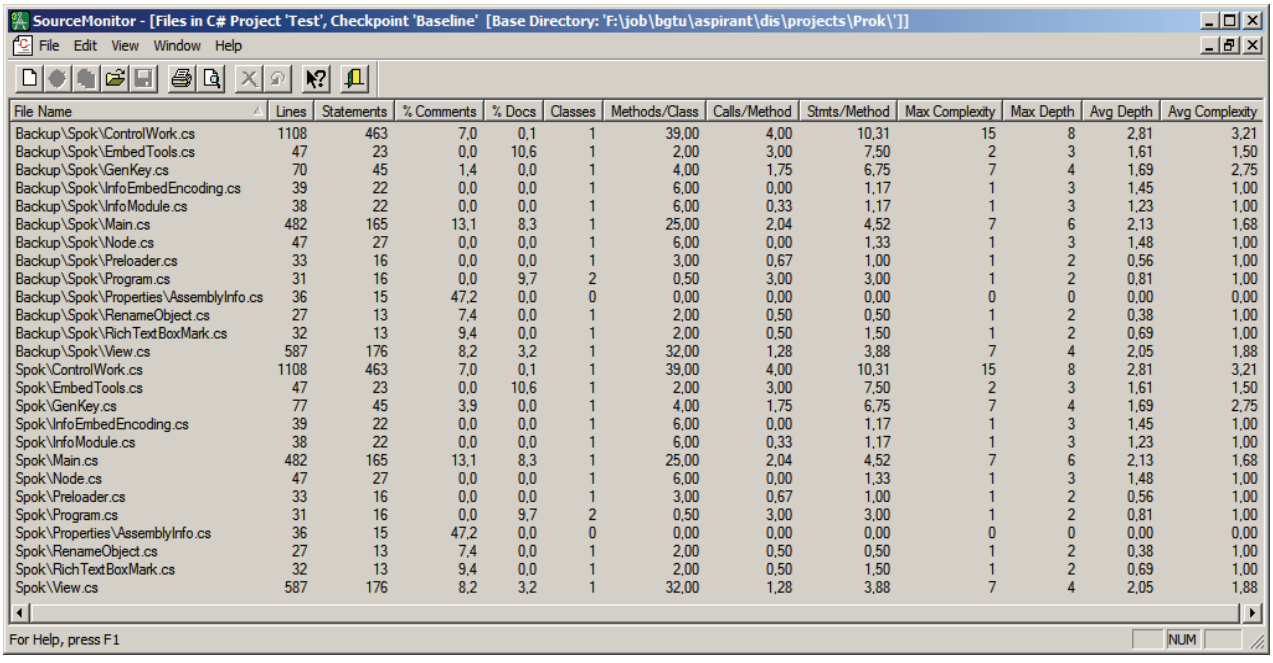


Рис. 3. Пример связи метрик
с графической схемой программы
с помощью инструмента Rational ClearCase [5]

SourceMonitor – инструмент для сбора метрик кода, написанного на C++, C, C#, VB.NET, Java, Delphi, Visual Basic (рис. 4).



File Name	Lines	Statements	% Comments	% Docs	Classes	Methods/Class	Calls/Method	Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Backup\Spok\ControlWork.cs	1108	463	7,0	0,1	1	39,00	4,00	10,31	15	8	2,81	3,21
Backup\Spok\EmbedTools.cs	47	23	0,0	10,6	1	2,00	3,00	7,50	2	3	1,61	1,50
Backup\Spok\GenKey.cs	70	45	1,4	0,0	1	4,00	1,75	6,75	7	4	1,69	2,75
Backup\Spok\InfoEmbedEncoding.cs	39	22	0,0	0,0	1	6,00	0,00	1,17	1	3	1,45	1,00
Backup\Spok\InfoModule.cs	38	22	0,0	0,0	1	6,00	0,33	1,17	1	3	1,23	1,00
Backup\Spok\Main.cs	482	165	13,1	8,3	1	25,00	2,04	4,52	7	6	2,13	1,68
Backup\Spok\Node.cs	47	27	0,0	0,0	1	6,00	0,00	1,33	1	3	1,48	1,00
Backup\Spok\Preloader.cs	33	16	0,0	0,0	1	3,00	0,67	1,00	1	2	0,56	1,00
Backup\Spok\Program.cs	31	16	0,0	9,7	2	0,50	3,00	3,00	1	2	0,81	1,00
Backup\Spok\Properties\AssemblyInfo.cs	36	15	47,2	0,0	0	0,00	0,00	0,00	0	0	0,00	0,00
Backup\Spok\RenameObject.cs	27	13	7,4	0,0	1	2,00	0,50	0,50	1	2	0,38	1,00
Backup\Spok\RichTextBoxMark.cs	32	13	9,4	0,0	1	2,00	0,50	1,50	1	2	0,69	1,00
Backup\Spok\View.cs	587	176	8,2	3,2	1	32,00	1,28	3,88	7	4	2,05	1,88
Spok\ControlWork.cs	1108	463	7,0	0,1	1	39,00	4,00	10,31	15	8	2,81	3,21
Spok\EmbedTools.cs	47	23	0,0	10,6	1	2,00	3,00	7,50	2	3	1,61	1,50
Spok\GenKey.cs	77	45	3,9	0,0	1	4,00	1,75	6,75	7	4	1,69	2,75
Spok\InfoEmbedEncoding.cs	39	22	0,0	0,0	1	6,00	0,00	1,17	1	3	1,45	1,00
Spok\InfoModule.cs	38	22	0,0	0,0	1	6,00	0,33	1,17	1	3	1,23	1,00
Spok\Main.cs	482	165	13,1	8,3	1	25,00	2,04	4,52	7	6	2,13	1,68
Spok\Node.cs	47	27	0,0	0,0	1	6,00	0,00	1,33	1	3	1,48	1,00
Spok\Preloader.cs	33	16	0,0	0,0	1	3,00	0,67	1,00	1	2	0,56	1,00
Spok\Program.cs	31	16	0,0	9,7	2	0,50	3,00	3,00	1	2	0,81	1,00
Spok\Properties\AssemblyInfo.cs	36	15	47,2	0,0	0	0,00	0,00	0,00	0	0	0,00	0,00
Spok\RenameObject.cs	27	13	7,4	0,0	1	2,00	0,50	0,50	1	2	0,38	1,00
Spok\RichTextBoxMark.cs	32	13	9,4	0,0	1	2,00	0,50	1,50	1	2	0,69	1,00
Spok\View.cs	587	176	8,2	3,2	1	32,00	1,28	3,88	7	4	2,05	1,88

Рис. 4. Сбор данных о метриках программного кода приложения «Prok»
с помощью инструмента SourceMonitor

Также с помощью SourceMonitor можно увидеть соотношение показателей метрик в виде диаграмм (рис. 5).

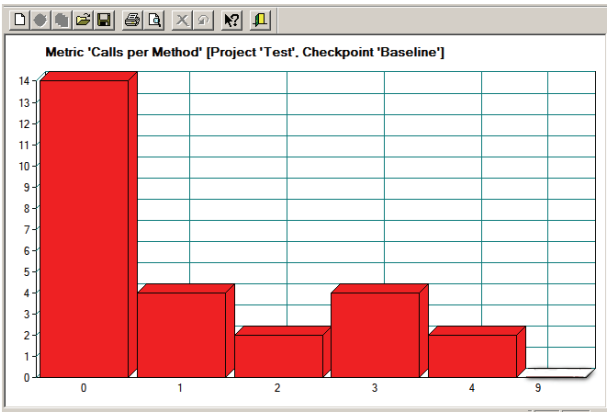


Рис. 5. Диаграмма вызовов методов в одном из классов приложения «Prok», построенная с помощью инструмента SourceMonitor

Для приложений, разработанных на языке Java, существует специальный инструмент JHawk, который, помимо стандартных количественных метрик, также подсчитывает метрики усилия, объема, и уязвимости Холстеда (рис. 6).

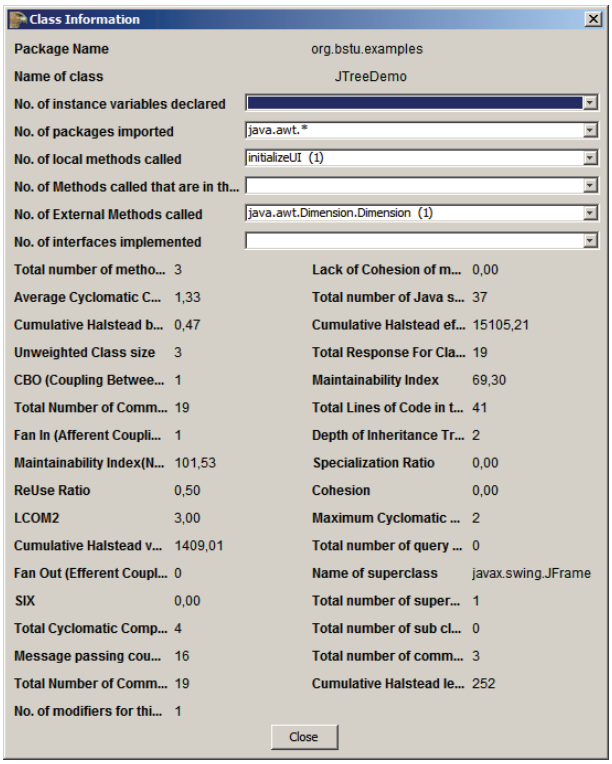


Рис. 6. Подсчет метрик Java-кода с помощью инструмента JHawk

CCCC (C and C++ Code Counter) – консольный инструмент, собирающий метрики. Результат его работы продемонстрирован на рис. 7.

Metric	Tag	Overall	Per Module
Number of modules	NOM	1	
Lines of Code	LOC	91	91.000
McCabe's Cyclomatic Number	MVG	16	16.000
Lines of Comment	COM	11	11.000
LOC/COM	L_C	8.273	
MVG/COM	M_C	1.455	
Information Flow measure (inclusive)	IF4	0	0.000
Information Flow measure (visible)	IF4v	0	0.000
Information Flow measure (concrete)	IF4c	0	0.000
Lines of Code rejected by parser	REJ	2	

Рис. 7. Сбор метрик с помощью утилиты CCCC на примере лабораторной работы по дисциплине «Информационная безопасность»

Среди которых метрика сложности Маккейба, а также метрики Чидамбера – Кемерера. Инструмент может анализировать коды C++ и Java.

Заключение. Существует множество различных метрик, но, к сожалению, многие из них либо описаны слишком абстрактно для практической реализации, либо не отвечают требованиям современных ООП приложений. Серьезную проблему также представляет их жесткая привязка к синтаксису языка. Тем не менее, существует ряд инструментов, позволяющих упростить этот анализ. А метрики, оценивающие процедурный код, могут быть использованы для анализа отдельных участков кода.

Литература

1. Ярмолик, В. М. Криптография, стеганография и охрана авторского права. Монография / В. Н. Ярмолик, С. С. Портянко, С. В. Ярмолик. – Минск, 2007. – 240 с.

2. Пласковицкий, В. А. Защита программного обеспечения от несанкционированного использования и модификации методами обфускации / В. А. Пласковицкий, П. П. Урбанович // Труды БГТУ. – 2011. – № 6: Физ.-мат. науки и информатика. – С. 173–176.

3. Практические исследования в области расчета метрик PVS-Studio [Электронный ресурс]. – 2010. – Режим доступа: http://www.viva64.com/ru/b/0014. – Дата доступа: 20.02.2013.

4. Martin, R. C. Designing Object-Oriented C++ Applications Using the Booch Method / R. C. Martin. – PrenticeHall, 1995. – P. 528.

5. Практическая реализация применения метрических показателей в IBM Rational ClearCase [Электронный ресурс]. – 2008. – Режим доступа: http://www.ibm.com/developerworks/ ru/edu/0108novich/section3.html. – Дата доступа: 20.02.2013.

Поступила 05.03.2013