Статистический анализ метрик программного кода

Е.Л.Романов, Л.А.Коршикова,

ФГБОУ ВО «Новосибирский государственный технический университет», Новосибирск, Россия. 630090, просп. К. Маркса, д. 20.

Аннотация — Использование метрик программного кода в управлении процессами программной инженерии затруднено в связи с отсутствием соответствующей отраслевой статистики. Описана программа сбора и анализа метрик байт-кода, позволяющая скачивать тематические выборки jar-файлов с репозитория maven, вычислять основные параметры и визуализировать гистограммы распределения метрик байт-кода. С помощью программы и стандартных средств статистического анализа проведено исследование статистики 11 метрик объектно-ориентированного кода на различных выборках jar-файлов. Определены объемные показатели выборки, на которой обеспечивается статистическая достоверность полученных законов распределения. Определены законы распределения и основные статистические параметры метрик. Приведены примеры экспертной оценки кода программного проекта на основе анализа метрик, нормированных к полученным статистическим значениям.

Ключевые слова — Java, байт-код, метрики программного кода, репозирорий, статистика, закон распределения, достоверность, отраслевая статистика, экспертная оценка качества проекта.

ВВЕДЕНИЕ

Программный код — единственная «материальная» ценность, создаваемая в программном проекте. От его качества зависят как текущие характеристики программного продукта, так и перспективы его успешного сопровождения. Качество кода определяется как соответствие формальным требованиям, выраженных в метрических характеристиках. Общепризнанный набор метрик в настоящее время «канонизирован» и не претерпевает существенных изменений. Существуют онлайн- и оффлайн-системы [1,2] документирования, сопровождения и получения метрических характеристик проекта, а также многочисленные плагины в интегрированных средах разработки [3]. На самой простой количественной метрике — количестве строк кода исходного текста (SLOC) — основана наиболее популярная методика оценки трудоемкости разработки программного обеспечения COCOMO [4].

Поддержка качества кода *не гарантирует его эксплуатационных свойств* – надежности, устойчивости, вероятности ошибок: перечисленные свойства обеспечиваются тестированием кода и артефактов проектирования. Однако при прочих равных условиях качество кода также способствует этому. Иными словами, некачественный код может быть сколь угодно «хорошим», но для разработки «хорошего» кода желательно поддерживать его качество.

Другим фактором не в пользу метрик является отсутствие рамочных законов, позволяющих оценить или рассчитать фундаментальные свойства программного продукта на основе его кода. Для сравнения: в мостостроении конструкция моста может быть сколь угодно оригинальной, но она рассчитывается по правилам и законам сопротивления материалов.

При существующем положении вещей получение метрических характеристик качества кода преследует цели:

- поддержание стандартов кода при коллективном владении кодом;
- обнаружение потенциально опасного кода;
- обнаружение узких мест в структуре кода;
- мониторинг разрабатываемого проекта в системе контроля качества при наличии статистики, развернутой по времени.

На практике в основном преобладает прагматический подход: используются средства поддержания стандартов кодирования – единообразная стилистика кода и простые количественные метрики.

Сами по себе метрические характеристики ничего не дают. Их необходимо как-то интерпретировать, как минимум, в сравнении с имеющейся статистикой. Здесь в программной инженерии сложилась парадоксальная ситуация. С одной стороны, нас окружает огромное количество общедоступного программного кода, как в исходных текстах, так и в промежуточных платформенно-независимых форматах (байт-код виртуальной машины Java - JVM). С другой стороны, общедоступная статистика его метрик отсутствует. Такая статистика по основным показателям качества программного кода, пусть даже качества формального, может быть полезна, прежде всего, как основа отраслевой статистики, относительно которой можно будет проводить исследования и оценки качества отдельных проектов. Пока что такая оценка выглядит как «вещь в себе» - полученные метрики не с чем сравнивать, можно только анализировать временной тренд полученных показателей.

Львиная доля общедоступного промежуточного кода представлена байт-кодом JVM, который является платформенно-независимым стандартом

де-факто. Средства распределенной сборки проектов и доступа к артефактам позволяют формализовать процесс получения необходимого статистического материала. На сегодняшний день один лишь репозиторий maven [7] хранит порядка 10^7 ссылок на артефакты (jar-файлы). Основным недостатком здесь является подверженность результатов измерения метрик оптимизациям компилятора.

МЕТРИКИ КОДА: ГДЕ, КАК И ЧЕМ СОБИРАТЬ?

«Исходный материал» для получения метрик доступен в трех формах: исходные тексты программ, промежуточный код и архитектурно-зависимый машинный код.

Исходные тексты программ позволяют получить наиболее точные показатели метрик. Статистическое исследование является репрезентативным в рамках конкретного языка программирования, а выборка представляет собой множество программ с открытым исходным текстом, реализованных на данном языке. В случае анализа интерпретируемых языков программирования это - единственный доступный подход.

Исследование машинного кода невозможно без привязки к аппаратной среде. В силу своего низкого уровня, оно ограничено в наборе метрик — так, невозможно провести анализ метрик объектно-ориентированного кода (объекты являются абстракцией высокого уровня). Выборка репрезентативна для программ, скомпилированных для определенного набора процессорных инструкций. В дополнении к традиционным метрикам здесь можно оценить время выполнения программ и объем потребления аппаратных ресурсов.

В байт-коде JVM сохраняется информация о сущностях и отношениях высокого уровня — таких как «Класс», «Метод», «Поле», «Вызов метода», и других, что позволяет измерить показатели метрик объектно-ориентированного кода. Большая часть оптимизаций выполняется во время выполнения программы, что позволяет избежать искажения показателей метрик. Выборка является платформенно-независимой, репрезентативной для семейства языков, компилируемых в байт-код JVM (более 200 языков), а также обладает наибольшим объемом по сравнению с двумя другими подходами.

На нижнем уровне системы сбора и анализа статистики необходимы средства, позволяющие анализировать байт-код и получать необходимые метрики. В *Таблице* 1 представлен список таких программных средств.

Представленные в списке решения не удовлетворяют необходимому набору требований: наличие свободной лицензии, актуальная поддержка, возможность массового исследования кода программы – одновременный анализ группы классов пакета или архива. Поэтому была проведена разработка оригинальной библиотеки сбора метрик байт-кода [8] на основе фреймворка ASM [9], который осуществляет разбор class-файлов через программный интерфейс, реализующий паттерн «посетитель».

Табл. 1.

ПО анализа метрик байт-кода

Название	Лицензия		Массовый анализ
Dependency Finder	Свободная (BSD License 2.0)	2010	Да
JDepend	Свободная (BSD License)	2005	Да
CyVis	Свободная (GNU General Public License)	2006	Да
Ckjm	Свободная (Apache License)	2014	Нет
Jqassistant	Свободная (GPLv3 License)	2018	Нет
JArchitect	проприетарная	2018	Да
Jtest	проприетарная	2018	Да
Sonargraph	проприетарная	2018	Да

ПРОГРАММА СБОРА, АНАЛИЗА И ВИЗУАЛИЗАЦИИ МЕТРИЧЕСКИХ ДАННЫХ JAVA-КОДА

Программа статистического анализа метрик байт-кода (рис.1,2) дает необходимый минимум средств для сбора, статистической обработки и разведочного анализа данных — метрик программного кода:

- сбор основных метрических характеристик байт-кода из проектов и библиотек, представленных в виде jar-файлов;
- поддержка выборок jar-файлов в одноименных каталогах;
- сканирование и загрузка jar-файлов с Maven-репозитория [5] по категориям, вводимым в виде ключевых слов поиска на сайте репозитория;
- сбор исходных данных метрик как по отдельным проектам/библиотекам, так и по категории в целом;
- генерация представительных выборок по исходному множеству значений метрики;
- экспорт основных характеристик законов распределения и данных гистограмм собираемых метрик в текстовые файлы;
- визуализация гистограмм распределений метрик.

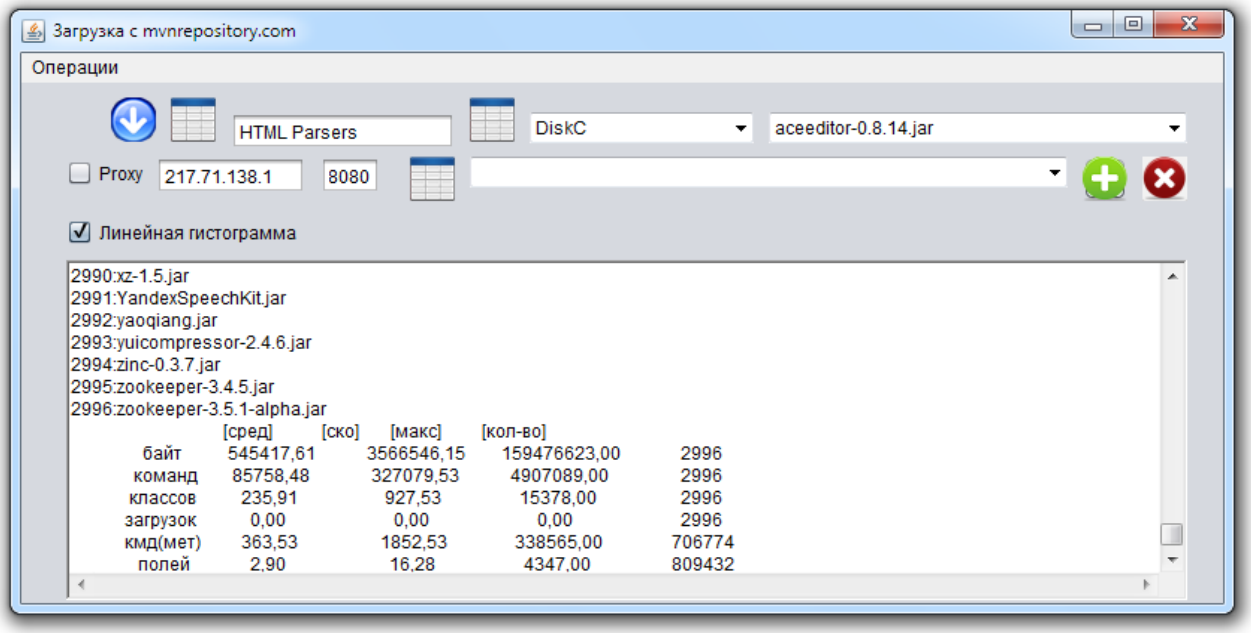

Рис. 1. Внешний вид программы

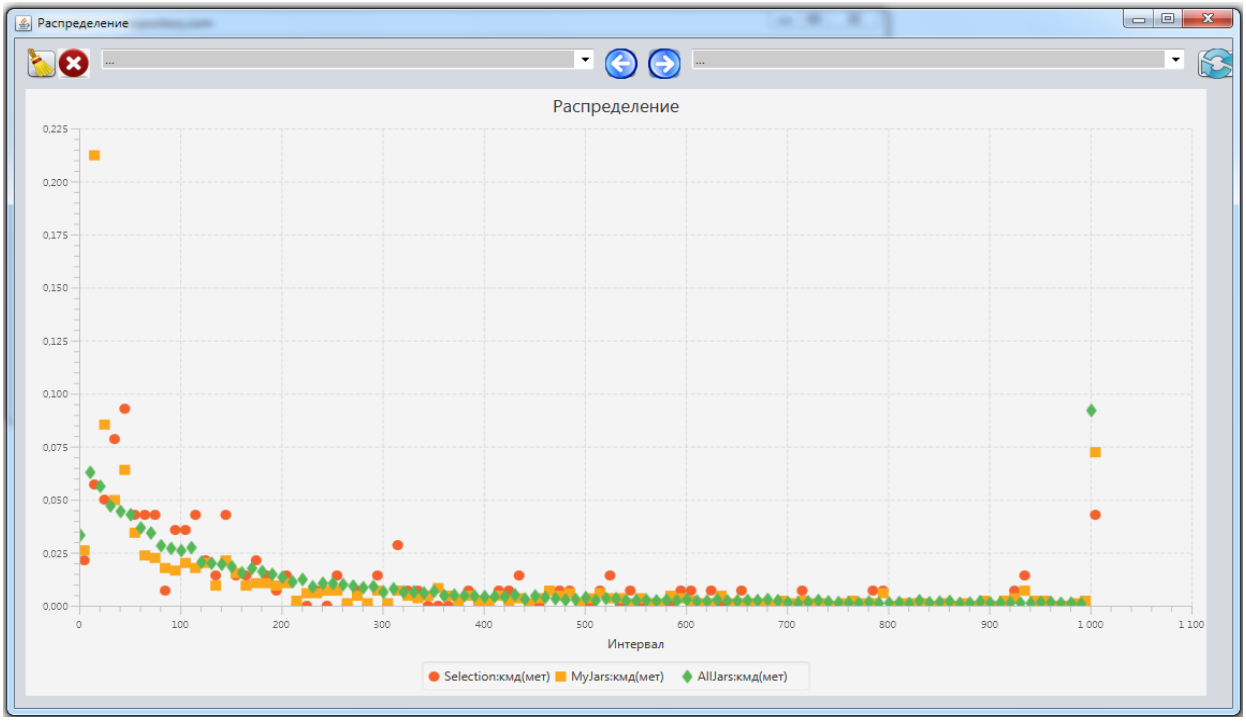

Рис.2. Гистограммы статистик метрики для разных категорий

АНАЛИЗ СТАТИСТИКИ БАЙТ-КОДА

Для практического использования статистики данных по метрикам байт-кода необходимо решить следующие задачи:

- определить, насколько объем исходных данных позволяет судить о достоверности вероятностных характеристик метрик – законов распределения и их основных параметров;
- произвести детальный анализ каждой метрики. Определить виды законов распределения данных метрики;
- разработать методику оценки состояния программного проекта на основе сравнения метрик проекта со статистикой метрик.

Метрики кода и исходные данные

Поскольку байт-код является объектно-ориентированным, статистической единицей

измерения метрики как случайной величины является класс.

Исследование проводилось для метрик двух видов. К первому относятся метрики байт-кода и проекта, не относящиеся к канонически наборам. Некоторые из них имеют отношение к проекту (jar-файлу) в целом. Все они имеют оригинальные аббревиатуры:

- размерность jar-файла проекта в байтах - **JFS** (JarFile Size);
- количество команд байт-кода в jar-файле (проекте) – **JCS** (Jar Code Size);
- количество загрузок проекта из репозитория – **NoDL** (Number of DownLoads);
- количество классов – **JNoC (**Jar - Number of Classes);
- количество полей в классе – **NoF** (Number of Filelds);
- количество методов в классе – **NoM** (Number of Methods);
- общее количество полей и методов класса – **NoFM** (Number of Fields and Methods);
- длина байт-кода метода, количество команд в методе – **MCS** (Method Code Size).

Следующие метрики из групп Чидамбера/Кемерера и Мартина являются стандартными и имеют общепринятые обозначения:

- центростремительное сцепление **Ca** (Afferent Couplings) – количество классов вне категории (пакета), зависящих от классов этой категории;
- центробежное сцепление **Ce** (Efferent Couplings) - количество классов внутри категории (пакета), которые зависят от классов вне этой категории:
- Нестабильность **I** (Instability), **I = Ce / (Ca+Ce)**;
- высота дерева наследования - **DIT** (Depth of Inheritance Tree) - максимальная длина пути (количество вершин – классов) по дереву наследования
- количество потомков класса - **NOC** (Number of Children) – среднее количество прямых наследований класса
- сцепление между классами - **CBO** (Coupling Between Object Classes) – общее количество вызовов методов и использования свойств объектов других классов в коде класса
- взвешенный суммарный вес методов в классе - **WMC** (Weighted Methods per Class) – сумма значений цикломатической сложности методов в классе

В качестве исходных данных использовались наборы jar-файлов, разбитые на группы в зависимости от цели анализа и «происхождения»:

Группа наборов большого объема (группа 1):

- все доступные файлы, скачанные из репозиториев, а также находящиеся на системном диске в различных средах разработки (Android, Java EE, Java SE, JDK, Scala, NetBeans, IntelJJ IDEA) – **DiskC (**2996 проектов, 806525 классов**);**
- jar-файлы большого размера (более 6.8 Мб) из первого набора;
- jar-файлы малого размера (менее 1 Мб) из первого набора.

Группа наборов по предметной области программирования, скачанных с репозитория Maven [5] – **HTTP** (категории HTTP Clients, HTTP Parser), **JDBC, JVM** (категории JVM Libraries, ByteCode Analysers), **JSON**, **Audio** (группа 2).

Отдельные jar-файлы большого размера распространенных средств разработки (**Android, RxJava**), набор jar-файлов собственных проектов кафедры (**MyJars**), отдельный файл собственного проекта (**BRSCore**) (группа 3).

Оценка собранных данных

Уже внешний вид гистограмм распределения значений метрики MCS, выбранной в качестве образца, позволяет судить о качественной разнице собранных данных для наборов разных групп (рис.3-5)

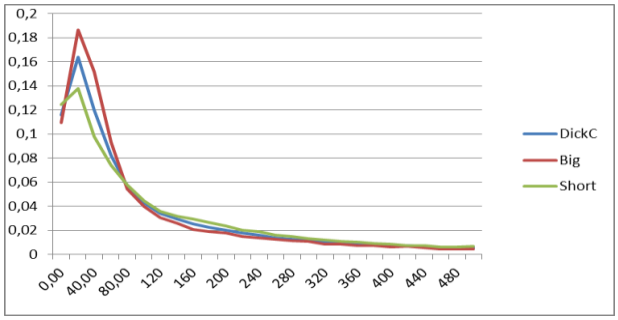

Рис. 3. Распределение метрики MCS для наборов группы 1

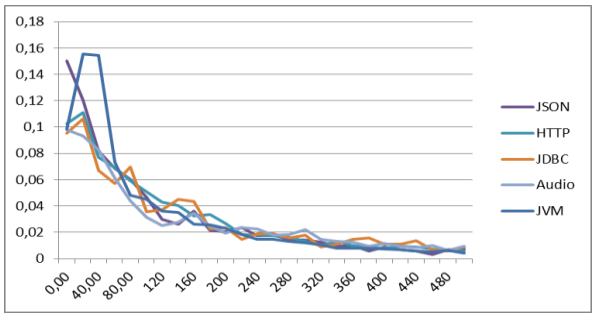

Рис. 4. Распределение метрики MCS для наборов группы 2

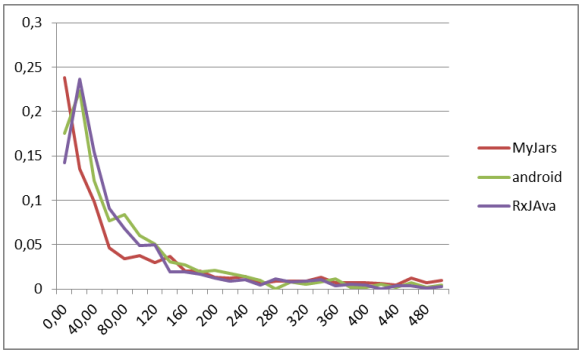

Рис. 5. Распределение метрики MCS для наборов группы 3

Определение соответствия выборки закону распределения является задачей *статистической проверки гипотезы* об ее принадлежности этому

распределению. Для решения этой и подобных задач использовался пакет статистического анализа EasyFit [8].

Поскольку вид большинства распределений выглядит как обратно экспоненциальный, «хвост» распределения создает статистический «шум» на малых частотах появления соответствующих значений метрики. Действующий диапазон значений метрики (рис.6) сокращается до размера, при котором значение *критерия оценки достоверности* превышает теоретическое значение для выбранного *уровня значимости (УЗ) гипотезы α,* т.е. закон устойчиво определяется. В нашем случае используется наиболее распространенный критерий Колмогорова-Смирнова.

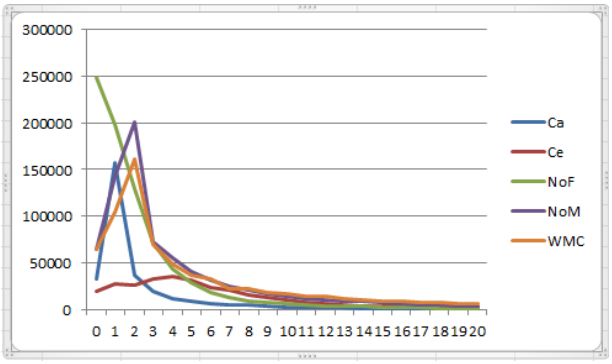

Рис. 6. Распределения метрик в диапазоне 0…20

В то же время сокращение диапазона увеличивает погрешность определения параметров распределения. В *Таблице* 2 для метрики NoM показано, как параметр экспоненциального распределения λ зависит от изменения диапазона.

Табл. 2.

Оценка диапазона анализируемых значений метрики NoM

λ	0,342	0,251	0,217	0,199	0,186
Критерий*	0,196	0,148	0,018	0,193	**0,204**
P-Value	0,756	0,695	0,243	0,077	0,02
Отвергнуто	-	-	-	-	+
Критерий**	0,391	0,287	0,234	0,205	**0,187**
Диапазон	11	21	31	41	51

Критерий* - расчетное значение критической статистики для критерия Колмогорова-Смирнова

Критерий** - теоретическое значение критической статистики при УЗ = 0,05

Имеет место естественная граница диапазона значений метрики, учитываемых в законе распределения. Она связана с условием принятия гипотезы – значение критерия не превышает критическое значение, установленное для выбранного уровня значимости.

Хотя большинство метрик имеют целочисленные значения, законы распределения определяются для непрерывных случайных величин ввиду разнообразия последних.

В таблице 3 приведены результаты определения пакетом EasyFit законов распределения для различных наборов для статистических данных метрики MCS – количество команд байт-кода в методе.

Табл. 3.

Законы распределения для метрики MCS

Метрика MCS	DiskC	Big	Short	JSON	JVM	HTTP	Audio	JDBC	MyJars	Android	RxJava	BRSCore
μ статистики	364	337	364	389	443	427	**840**	548	327	**106**	142	546
σ статистики	1856	2035	1333	1170	2932	1570	**3541**	1385	856	**166**	342	1340
Классов (NC)	806525	313208	286362	7532	26380	6171	1723	2666	843	985	931	103
Log_{10}(NC)	5,91	5,50	5,46	3,88	4,42	3,79	3,24	3,43	2,93	2,99	2,97	2,01
Критерий*	0,037	0,036	0,036	0,036	0,059	0,032	0,037	0,041	0,115	0,038	0,051	0,070
P-Value	0,993	0,995	0,994	0,995	0,752	0,999	0,993	0,986	0,067	0,997	0,846	0,904
Критерий**	0,121	0,121	0,121	0,121	0,121	0,121	0,121	0,121	0,121	0,134	0,116	0,170
Распределение	1	1	2	2	1	3	2	3	4	2	1	4
σ	1,14	1,08	1,20	1,17	1,06	1,15	**1,16**	1,13	1,13	**1,15**	1,12	1,14
μ	4,15	4,09	4,21	4,26	4,22	4,36	**4,5**	4,49	4,26	**3,97**	4,01	4,28
Jar-файлов	3356	53	3082	132	64	51	**46**	291	3	**1**	1	1

Распределение: 1-логнормальное, 2-Джонсона, 3-экспоненциальное, 4-Парето

На *Рис*. 7 изображены зависимости значения критерия от размерности выборки - значений метрики в наборе (количества классов, для которых определено MCS). Ось X –десятичный порядок размерности выборки.

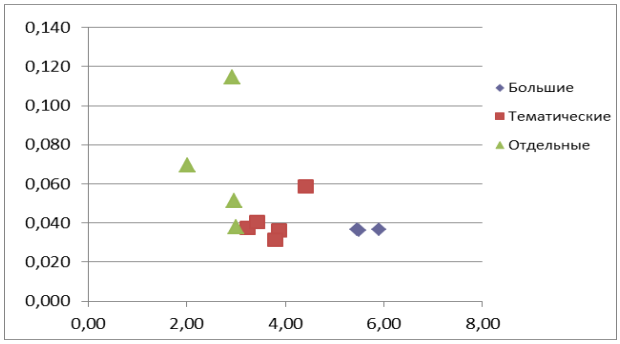

Рис. 7 . Значение критерия Колмогорова-Смирнова в зависимости от порядка выборки

Большинство наборов, в том числе состоящих из отдельных jar-файлов, позволяют «установить» закон распределения с учетом вероятностного характера проводимой процедуры. Значения критерия лежат в узком диапазоне для всех наборов групп 1 и 2.

В то же время основные параметры статистики (среднее и среднеквадратичное отклонение) меняются в широких пределах на различных наборах (рис.8). Одновременно меняются и параметры закона распределения. (σ и μ для логнормального распределения – в табл. 3). Между теми и другими наблюдается естественная корреляция.

«Выбросы» параметров для тематических выборок и отдельных jar-файлов вполне могут быть объяснены спецификой предметной области. Например, малое среднее MCS для Android и RxJava – преобладанием коротких методов при высоком уровне модульности и абстрагирования в коде, а значительное среднее в Audio – наличием объемных методов обработки аудио-данных.

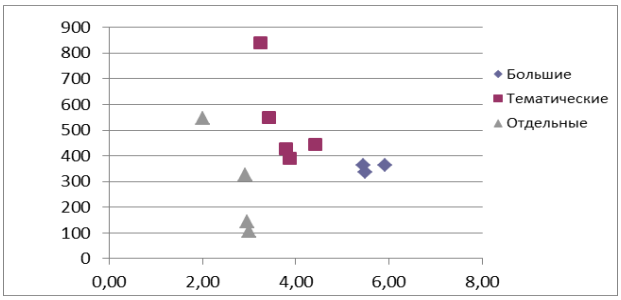

Рис. 8. Среднее значение метрики MCS на различных наборах

Таким образом, только наборы группы 1 могут быть использованы в качестве статистического материала для получения статистических параметров и законов распределения метрик байт-кода. Размерность исходных данных должна быть не менее 10^5 классов.

Законы распределения метрик

Как было показано, статистический анализ метрик проводится на выборке группы 1 (DiskC). Данные анализа метрик приведены в таблице 3.

Табл. 4.

Итоговые параметры оценки законов распределения для различных метрик

Метрика	MCS	WMC	CBO	DIT	NoC	NoF	NoM	I	NoFM	Ce	Ca	JNoC	JCS	JFS
Критерий*	0,037	0,128	0,07	0,262	0,069	0,240	0,148	0,120	0,084	0,081	0,024	0,047	0,069	0,041
P-Value	0,993	0,839	0,94	0,215	0,999	0,151	0,695	0,418	0,996	0,997	0,130	0,973	0,984	0,999
Критерий**	0,121	0,287	0,19	0,338	0,287	0,287	0,287	0,187	0,287	0,287	0,28	0,134	0,21	0,154
Распределение	4	2	4	3	7	3	6	1	2	5	3	6	5	2
Ячеек гистограммы	126	21	51	15	21	21	21	51	21	21	22	100	40	75
Шаг выборки	4	1	1	1	1	1	1	0	1	1	1	1	400	2000
Диапазон	504	21	51	15	21	21	21	1,02	21	21	22	100	16000	2E+05
Ранг распределения	1	1	1	1	1	1	4	2	1	1	1	2	1	3

Распределение: 1-бета,2-гамма,3-Гумбеля,4-логнормальное,5-Джонсона,6-экспоненциальное,7-степенное

Результаты можно сгруппировать следующим образом. К первой группе относится большая часть метрик (MCS,WMC,CBO,NoM, NoFM, Ce,Ca,JFS), которая имеет характерный вид кривой распределения (*Рис*. 9).

Для большинства метрик характер такого распределения легко объясним. Например, для суммарной цикломатической сложности (рис.10) первый отсчет соответствует часто встречающимся классам (интерфейсам) с пустыми методами, классы с линейным кодом в методах также встречаются часто и имеют малые значения метрики

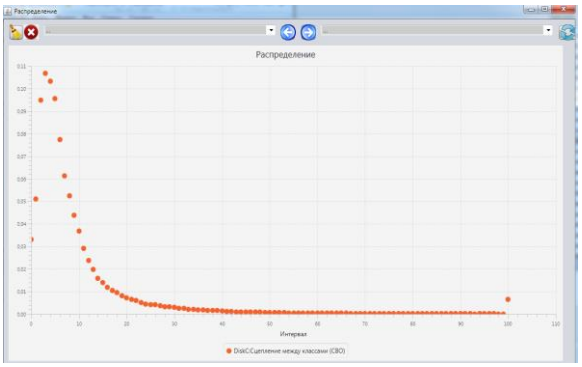

Рис. 9. Вид эмпирической кривой распределения первой группы (CBO)

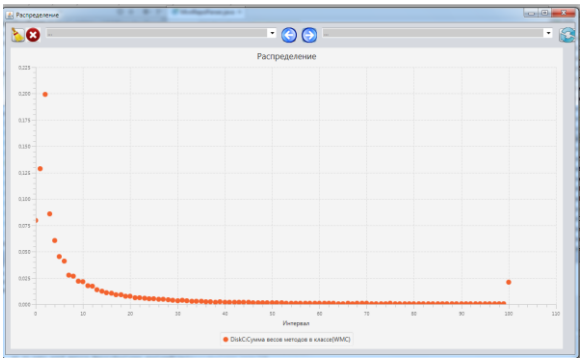

Рис. 10. Статистика метрики – суммарная сложность кода класса (WMC)

Характерный убывающий вид графика соответствует общим принципам модульного программирования и ООП – преимущественное использование «коротких» и простых сущностей (классов, методов) перед сложными и объемными. В данном случае статистика подтверждает, что разработчики в своей основной массе следуют этому правилу.

Вторая группа метрик (DIT, JCS, JNoC, NoC, NoF) отличается отсутствием подъема в начальной точке кривой распределения (NoF на рис.11). Для указанной метрики это обусловлено большим количеством классов без данных – интерфейсов, классов со статическими методами, а также классов с минимальным количеством данных (полей).

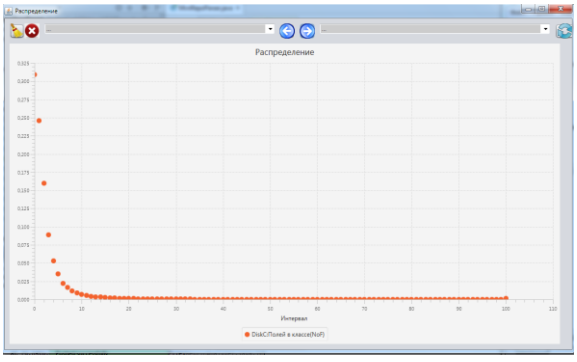

Рис. 11. Статистика метрики – количество полей в классе» (NoF)

Статистика метрики «нестабильность» имеет особый вид распределения (рис.12) объясняется тем, что метрика вычисляется как **I = Ce / (Ca+Ce)**, что при большой частоте малых значений Ca и Ce дает выбросы на значениях 0.25, 0.33, 0.4, 0.5 и т.п.

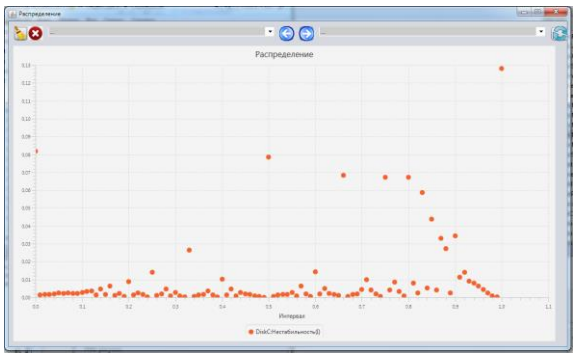

Рис. 12. Статистика метрики – нестабильность (I)

ОЦЕНКА СОСТОЯНИЯ ПРОЕКТА

Статистические данные для оценки состояния проекта нельзя использовать формально, необходим содержательный анализ специфики проекта. Это объясняется самой природой метрик программного кода: сами по себе значения метрик не являются показателем качества проекта. Наличие достоверной статистики дает дополнительные основания для интерпретации и оценки результатов измерений. Возможно несколько типовых вариантов оценки расхождения метрик и статистики:

- отклонение отдельных метрик может объясняться спецификой проекта, это объяснение найдено и правдоподобно;
- массовое отклонение метрик может свидетельствовать о низком качестве проекта, отклонение отдельных метрик могут быть оценены как отступление от принципов технологии ООП;
- отклонение отдельных метрик обусловлено самой природой статистического анализа и не является показателем качества проекта.

Рассмотрим несколько вариантов анализа конкретных проектов (jar-файлов). Сравнение будем производить по основным параметрам статистик (МО, СКО) метрик проекта и выборки большого объема **DiscC**. Значения метрик анализируемого проекта нормированы к соответствующим значениям метрик в выборке. Для нормированных значений используются те же самые аббревиатуры, что и для исходных метрик.

Пример анализа. Библиотека ядра системы учета рейтинга успеваемости BRSCore

Библиотека (собственный проект кафедры) содержит общий программный код клиент-серверных приложений для системы учета рейтинга успеваемости– уровни доступа к данным, бизнес-уровень, коммуникации, контроллер представлений (экранных форм). Нормированные метрики проекта (рис.13) имеют МО в пределах 0.5-1.5 от статистических, что требует объяснения.

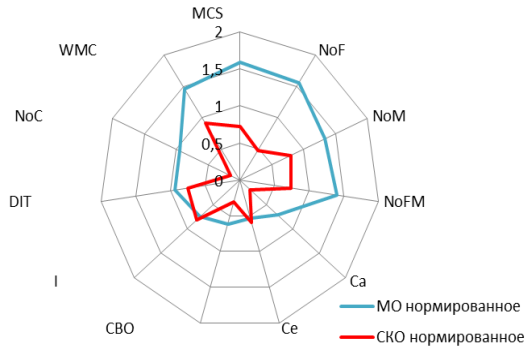

Рис. 13. Нормированные метрики проекта библиотеки BRSCore

Значительные превышение нормированных значений длины кода метода (MCS = 1.58), суммарной сложности методов классов (WMC = 1.46), количества полей (NoF = 1.56) и методов (NoM = 1.33) свидетельствуют о нестрогом соблюдении принципов ООП («раздутые» классы, отсутствие необходимых абстракций). Этим же могут объясняться заниженные значения сцепления (Ca, Ce, CBO) – методы классов «самодостаточны», каждый использует собственные средства.

Пример анализа. Библиотека реактивного программирования ReCore

Библиотека среды реактивного программирования ReCore (собственный проект кафедры) не содержит элементов интерфейса пользователя, реализует большой набор внутренних абстракций (реактивные типы данных, классы), каждая абстракция имеет достаточно большое количество реализаций-наследников. Библиотека имеет размерность 0.4 от среднестатистической. Нормированные метрики проекта (*Рис.* 14) имеют МО в пределах 0.8-1.2 от статистических, что свидетельствует об отсутствии каких-то явных пороков.

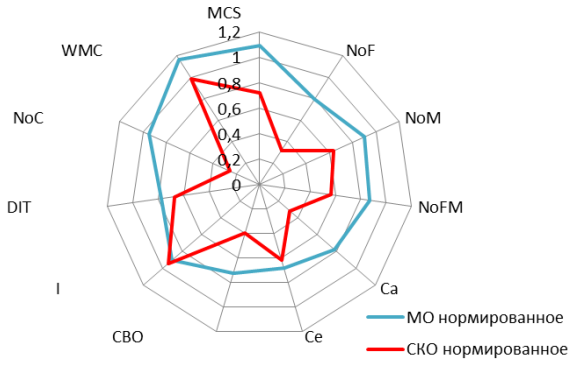

Рис. 14. Нормированные метрики проекта ReCore

Для отклонений нормированного СКО проекта можно найти разумные объяснения, связанные с особенностями проекта:

- NoF = 0.3 – однородность классов по количеству полей;
- NoC = 0.25 – однородность по количеству наследников, однотипность вариантов наследования;
- CBO = 0.4 – однородность по связям между классами;
- Ca = 0.3 - однородность по использованию различных пакетов проекта сторонними классам.

Возможно, в проекте имеет место повторяемость идентичных схем или групп классов, которые снижают разброс статистических данных.

Пример анализа. Библиотека среды исполнения программ в Android

Библиотека Android представляет собой библиотеку окружения (среды исполнения), в которой работает Java-программа в ОС Android. Фактически она реализует всю специфику среды функционирования программ в Android, ниже ее расположены библиотеки Linux и сам Linux. Библиотека имеет размерность кода, сопоставимую со среднестатистической, но значительное количество классов (730 против 235 в среднем). Вид нормированных метрик (*Рис*. 15) показывает, что они сильно отличаются от статистических.

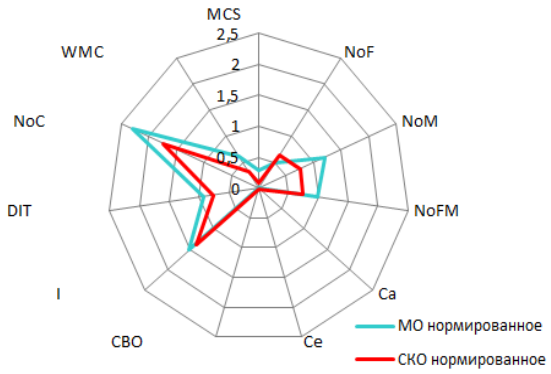

Рис. 15. Нормированные метрики библиотеки Android

Библиотека имеет очень низкие нормированные значения сцепления (Ca = 0.006, Ce = 0.004, CBO = 0.005), что объясняется высокой автономностью компонент и отсутствием прямых связей между классами из разных пакетов. Каждый пакет (категория) представляет собой отдельный сервис и самой библиотекой не используется.

Значительное количество потомков (NoC = 1,75) указывает на «популярность» использования базовых классов и абстракций, а малые значения нормированной длины метода (MCS = 0.08) и взвешенной сложности класса (WMC = 0.3) – об исключительном использовании коротких методов с простой логикой.

Пример анализа. Стандартная среда реактивного программирования RxJava

Среда реактивного программирования [7] предлагает средства для организации событийного управления на основе подписки на события и широко используется как средство организации программы на Java, в том числе взаимодействия внешних форм (графического интерфейса). Библиотека среднего размера – около 1 Мб. Вид

нормированных метрик (*Рис*. 16) показывает, что некоторые из них значительно отличаются от статистических.

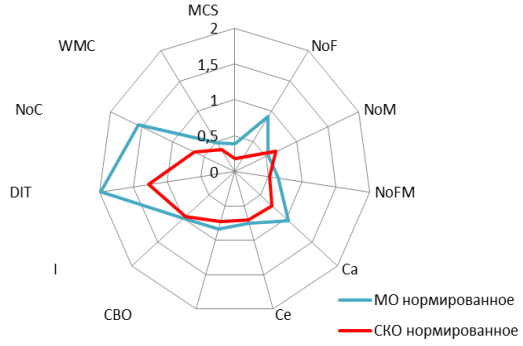

Рис. 16. Нормированные метрики библиотеки RxJava

Значительная глубина дерева наследования (DIT = 2), количество наследников (NoC = 1.55), малая длина методов (MCS = 0.4) и низкая суммарная сложность (WMC = 0.47) характеризуют разработку как исключительно модульную, с высоким уровнем наследования. Это согласуется с тем, библиотека поддерживает специфическую парадигму программирования, для которой необходимо создавать значительное количество абстракций и развивать их.

ВЫВОДЫ

Проведенные измерения и анализ их результатов показал, что многообразие окружающего нас программного кода может быть описано в том числе и средствами статистики через метрические характеристики этого кода. Статистика большинства метрик соответствует ограниченному числу законов распределения, гипотезы соответствия подтверждаются уже на тематических выборках порядка сотен jar-файлов и тысяч классов. Полученные данные могут быть использованы для экспертной оценки качества программных проектов на основе сопоставления их метрик с имеющейся статистикой.

ЛИТЕРАТУРА

[1] Облачный сервис CodeNforcer http://cloud.codenforcer.com.
[2] SonarQube: The leading product for continuous code quality https://www.sonarqube.org/.
[3] NetBeans: SourceCodeMetrics - plugin detail http://plugins.netbeans.org/plugin/42970/sourcecodemetrics.
[4] Boehm Barry, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000
[5] Maven Repository http://mvnrepository.com/repos
[6] EasyFit – Distribution Fitting Made Easy http://www.mathwave.com/.
[7] RxJava – Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observable sequences for the Java VM https://github.com/ReactiveX/RxJava.
[8] Васильев Н.Е. Исследование и разработка системы анализа метрик программного кода. Дни науки НГТУ–2017: материалы научной студенческой конференции. Новосибирск: Изд-во НГТУ, 2017. С. 23-25.
[9] ASM: A Java bytecode engineering library http://asm.ow2.io/.

Евгений Леонидович Романов, доцент, к.т.н., доцент кафедры вычислительной техники НГТУ.
E-mail: romanov@corp.nstu.ru

Лариса Александровна Коршикова, доцент, к.т.н., доцент кафедры вычислительной техники НГТУ.
E-mail: lk@vt.cs.nstu.ru

Статья поступила в редакцию 13 сентября 2018 г.

Statistical Analysis of Program Code Metrics

E.L. Romanov, L.A. Korshikova

Novosibirsk State Technical University, Novosibirsk, Russia. 630090, K. Marx, 20.

Abstract - The use of program code metrics in the management of software engineering processes is difficult due to the lack of relevant industry statistics. A program for collecting and analyzing bytecode metrics is described, which allows downloading thematic samples of jar files from the maven repository, calculating the basic parameters, and visualizing histograms for the distribution of bytecode metrics. Using the program and standard statistical analysis tools, the statistics of 11 metrics of object-oriented code were analyzed on different samples of jar-files. The volumetric indexes of the sample are determined, on which the statistical reliability of the obtained distribution laws is ensured. The distribution laws and the basic statistical parameters of metrics are determined. Examples of expert evaluation of the code of a software project based on the analysis of metrics normalized to the statistical values obtained are given.

Key words - Java, bytecode, program code metrics, repository, statistics, distribution law, reliability, industry statistics, expert evaluation of project quality.

REFERENCES

[1] Cloud service CodeNforcer http://cloud.codenforcer.com.
[2] SonarQube: The leading product for continuous code quality https://www.sonarqube.org/.

[3] NetBeans: SourceCodeMetrics - plugin detail http://plugins.netbeans.org/plugin/42970/sourcecodemetrics.
[4] Boehm Barry, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000
[5] Maven Repository http://mvnrepository.com/repos
[6] EasyFit – Distribution Fitting Made Easy http://www.mathwave.com/.
[7] RxJava – Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observable sequences for the Java VM https://github.com/ReactiveX/RxJava.
[8] Vasilyev N.E. Research and development of a system for analyzing program code metrics. Days of Science of the NSTU-2017: materials of a scientific student conference. Novosibirsk: Publishing house of the NSTU, 2017. S. 23-25.
[9] ASM: A Java bytecode engineering library http://asm.ow2.io/.

Evgeny Leonidovich Romanov, Associate Professor, Candidate of Technical Sciences, Associate Professor of the Computer Science Department of the NSTU.
E-mail: romanov@corp.nstu.ru

Larisa A. Korshikova, Associate Professor, Candidate of Technical Sciences, Associate Professor of the Computer Science Department of the NSTU.
E-mail: lk@vt.cs.nstu.ru

The article was received on September 13, 2018.