Метрики сложности кода

Технический отчет 2012-2

Ледовских Илья
27.01.2012

В отчете приводится обзор метрик сложности программного обеспечения применительно к задачам анализа программ в двоичных кодах. В обзор включены количественные метрики, метрики сложности потока управления, метрики сложности данных, комбинированные метрики, объектно-ориентированные и гибридные метрики; делаются выводы о применимости метрик этих групп к анализу двоичного кода. Рассматривается влияние на значения метрик различных видов запутывания программ – запутывания форматирования, запутывания данных, запутывания потока управления и превентивных трансформаций. Составлен список метрик, которые представляются наиболее подходящими для оценки сложности программ в двоичных кодах. В заключении делаются выводы о применимости различных метрик к задачам анализа запутанного кода, и формулируются задачи для дальнейших исследований.

Оглавление

I Введение 2
2. Метрики, применяемые при разработке ПО 5
2.1. Количественные метрики 5
2.2. Метрики сложности потока управления программы 7
2.3. Метрики сложности потока данных программы 12
2.4. Комбинированные метрики сложности управления и данных 14
2.5. Объектно-ориентированные метрики и метрики надежности 15
2.6. Гибридные метрики 15
3. Запутывающие преобразования и их влияние на известные метрики 16
3.1. Запутывание форматирования 16
3.2. Запутывание данных 17
3.3. Запутывание потока управления 18
3.4. Превентивные трансформации 20
4. Заключение 20
Список использованных источников 22

I Введение

Применение метрик программного кода позволяет разработчикам и руководителям проектов оценивать различные свойства создаваемого или уже существующего программного обеспечения, прогнозировать объем работ, давать количественную характеристику тех или иных проектных решений, качества разработанных систем и их частей, характеризовать сложность или надежность программного обеспечения. Типичный сценарий использования метрик при разработке и модификации программного обеспечения таков:

1. Для уже существующей программы в соответствии с принятыми для ее оценки метриками вычисляется набор мер, характеризующих ее сложность, трудоемкость разработки и анализа, стилистику, надежность и т.д. в соответствии с теми стандартами, которые приняты в данной компании или организации. Предполагается, что уже существующий код соответствует эталонным значениям используемого набора мер.

2. После внесения изменений в программу оценка ее метрических характеристик выполняется повторно. Новые значения мер сравниваются как с эталонными значениями, так и с характеристиками программы до модификации. Таким образом, не только выполняется проверка нового программного кода на соответствие стандартам разработчика, но и оценивается влияние изменений на ранее достигнутые показатели (сложности, надежности, понятности и т.д.) программного продукта.
3. Принимается решение о возможности и целесообразности включения нового кода в проект. Оцениваются трудозатраты и персональный вклад разработчиков на внесение изменений в программу.

Конечно, при разработке программ и оценке усилий разработчиков различные метрики (особенно количественные) могут служить скорее рекомендательными характеристиками, поскольку возможны разного рода ухищрения со стороны не вполне добросовестных разработчиков, с целью как минимизировать, так и максимизировать те или иные используемые меры. Классический пример связан с передачей ряда продуктов на аутсорсинг в Индию – после ввода оценки труда в виде количества строк кода этот показатель в одночасье удвоился. В то же время, малая количественная оценка может свидетельствовать не о недостаточности усилий, а о сложности выполняемой работы – как разработки или модификации, так и анализа программ. Например, поиск ошибки в критически важном фрагменте приложения может занять дни и недели, а исправление обнаруженной ошибки – свестись к одной-единственной строке кода. Последняя проблема может быть в значительной степени решена путем использования для оценки сложности поставленной перед разработчиком задачи метрик сложности вместо количественных метрик.

Задача анализа приложений в условиях отсутствия исходных текстов предъявляет несколько иные требования к выбору метрик кода, чем процесс разработки и модификации приложений. Ряд количественных метрик, в основном используемых для оценки трудозатрат по проектам, основан на характеристиках исходного кода. Самой элементарной из таких метрик является SLOC (Source Lines Of Code – количество строк кода) и ее варианты (среднее число строк для функций/классов/модулей). Также при оценке проектов используют количество пустых строк, количество или процент комментариев, оценку стилистики программы и метрики Холстеда. В исходном виде эти метрики не могут использоваться применительно к анализу двоичного кода; то же самое относится к некоторым метрикам сложности, также основанным на подсчёте конструкций исходного кода. Однако некоторые известные метрики, как количественные, так и сложности, могут быть модифицированы применительно к анализу

двоичных кодов, в частности, применительно к анализу трасс программ. Для определения перечня пригодных для анализа двоичного кода метрик и их модификаций необходимо сформулировать задачи, которые с помощью данных метрик планируется решать. Представляется, что должны быть выделены 3 группы задач для использования метрик программного обеспечения применительно к анализу приложений в двоичных кодах:

1. **Классификация приложений** по сложности анализа. Анализируемое приложение (его двоичный код или набор трасс) оценивается посредством некоторых метрик, затем полученные значения сопоставляются со шкалами ранее полученных эталонных значений для выбранных метрик, и приложение относится к тому или иному классу сложности/трудоемкости анализа. Помимо выбора набора метрик, для решения данной задачи важно собственно построение эталонных шкал, то есть решение задачи подразумевает сбор большого объема статистических данных по значительному числу различных анализируемых приложений.
2. **Оценка трудоемкости** анализа частей одного приложения. Применяя разнообразные метрики не к приложению целиком, а к его компонентам – модулям, функциям и нитям управления, можно оценить трудозатраты на анализ различных частей программы. Это позволит, прежде всего, оптимальным образом распределить задачу между участвующими в проекте аналитиками, и получить результаты по всему проекту в более сжатые сроки.
3. **Профилирование** приложения с целью автоматического выявления потенциально интересных для аналитика фрагментов кода. Говоря о бинарном коде приложения, мы практически всегда подразумеваем, что этот код защищен от анализа различными встроенными в него механизмами защиты. В силу причин, обусловленных влиянием различных видов защиты на эффективность выполнения кода, эти защитные механизмы применяются неоднородно. Наиболее мощными средствами защищаются критичные с точки зрения безопасности участки, время выполнения которых относительно невелико; важные, но критичные по времени счёта фрагменты защищаются с помощью менее затратных преобразований. Значительная часть приложения, не содержащая важного или уникального кода и не представляющая интереса для анализа, может быть вообще свободна от какой-либо защиты. Представляется целесообразным подобрать набор метрик, применимых как к программным единицам приложения (функциям и модулям), так и к произвольным его фрагментам различной длины (последовательностям инструкций листинга, трассам

выполнения и подтрассам выделенных алгоритмов). Эти метрики должны достаточно чётко характеризовать фрагменты кода по наличию в них запутывания или иных механизмов защиты, и по возможности точно локализовать защищенные в разной степени участки внутри анализируемых фрагментов. Применимость метрик к достаточно коротким фрагментам кода или трасс должна позволить строить профили сложности и защищенности анализируемого кода, позволяющие не только локализовать защитные средства в коде приложения, но и уточнять полученные ранее оценки сложности программных единиц приложения (то есть оперативно корректировать решение задач второй группы – получение оценок трудоемкости анализа частей приложения).

2.Метрики, применяемые при разработке ПО

2.1. Количественные метрики

Количественные характеристики программ обычно рассматриваются в первую очередь, ввиду их простоты. Традиционно эти метрики вычисляются на основе исходного кода программ, что по понятным причинам не является возможным при анализе бинарного кода приложений. Рассмотрим только те метрики, которые могут быть видоизменены применительно к двоичному коду анализируемой программы.

1. **Количество строк кода** (SLOC – Source Lines Of Code). На практике различают физические и логические строки (команды, операторы). Применительно к бинарному коду, эта метрика должна определяться количеством инструкций в ассемблерном листинге. Ценность этой меры применительно к анализируемому приложению невелика. Для построения профиля сложности может применяться такая вариация данной метрики, как число инструкций базового блока – чем мельче базовые блоки, тем «гуще» передачи управления внутри фрагмента кода; для того, чтобы данная мера возрастала с ростом сложности, необходимо брать обратную величину.
2. **Среднее число строк** для функций (классов, файлов, модулей). Для бинарного кода – среднее число инструкций в функции. Возможная вариация – средний размер базового блока в функции (для оценки сложности по увеличению меры – обратная величина).
3. **Метрики Холстеда** (Halstead) позволяют частично учесть возможность записи одной и той же функциональности разным количеством строк и операторов. Они основаны на большом числе количественных показателей, таких, как число

уникальных операторов программы, число уникальных операндов программы, общее число операторов, общее число операндов, теоретическое число уникальных операторов и теоретическое число уникальных операндов. Через эти показатели различными формулами определяются *уровень качества программирования*, *сложность понимания программы, трудоемкость кодирования программы, уровень языка выражения, информационное содержание программы* (данная характеристика позволяет определить умственные затраты на создание программы) и *оценка интеллектуальных усилий* при разработке программы, характеризующая число требуемых элементарных решений при написании программы. Данная метрика была изначально разработана для оценки качества программ на Фортране и трудоемкости их разработки; количественные показатели определяют словарь программы на фортраноподобном языке высокого уровня. Поэтому адаптация метрики Холстеда к бинарному коду приложения не представляется целесообразной.
4. **Метрики Джилба** (Jilb) показывают сложность программного обеспечения на основе насыщенности программы условными операторами или операторами цикла. CL – абсолютная сложность (количество управляющих операторов), cl=CL/n, где n – общее число операторов программы – относительная сложность программы по Джилбу. Несмотря на свою простоту, данная метрика хорошо отражает как трудоемкость разработки, так и сложность понимания программы. При добавлении показателя максимального уровня вложенности условных операторов и циклов, эффективность данной метрики значительно возрастает. Существует также ряд вариантов этой метрики (отдельно для циклов и условных операторов, для модулей и связей между ними, для ненормальных выходов из управляющих операторов и функций, и так далее). Простота подсчета метрик Джилба, гибкость в выборе оцениваемых конструкций, возможность повышения оценки за счет введения дополнительных показателей сложности, а также возможность вычисления относительной меры, делают данную метрику интересной для задач анализа бинарного кода приложений.
5. **ABC-метрика** (Fitzpatrick) основана на подсчете присваиваний значений переменным (**A**ssignment), явных передач управления за пределы области видимости, т.е. вызовов функций (**B**ranch), и логических проверок (**C**ondition). Мера записывается тройкой значений, например, ABC = <7, 4, 2>, но для оценки сложности программы вычисляется одно число, как квадратный корень из суммы квадратов A, B, C. Эта метрика легко вычисляется, может быть вычислена для разных фрагментов кода и наглядна

при визуализации (вектор в трехмерном пространстве). К числу ее недостатков относят тот факт, что она может иметь нулевое значение для некоторых непустых программных единиц.

Из перечисленных количественных метрик при анализе программ в двоичных кодах представляется целесообразным использовать модифицированные метрики Джилба и ABC-метрику. Модификация метрики Джилба сводится к простой замене при подсчете операторов программы на языке высокого уровня инструкциями передачи управления в двоичном коде программы. Дополнительное достоинство этой метрики – возможность использования оценки относительной сложности для построения профиля сложности трассы. Число инструкций базового блока также может учитываться при построении профиля; средний же размер блока фактически учитывается относительной метрикой Джилба.

ABC-метрика может быть применена для оценки размеров и сложности фрагментов анализируемого приложения, а также для автоматического поиска (по нулевому значению данной меры) тривиальных функций – заглушек или частей запутанного кода.

Метрики SLOC и среднее число строк в функции непригодны для оценки сложности, метрики Холстеда – с одной стороны, избыточны, а с другой – ориентированы на оценку достаточно больших программ и неприменимы к коротким фрагментам трасс, то есть не позволяют построить адекватный профиль сложности анализируемой программы.

2.2. Метрики сложности потока управления программы

Метрики данной группы, за исключением метрики Вудворда, основаны на анализе графа потока управления программы.

1. **Цикломатическая сложность**, или **метрика Мак-Кейба** (McCabe) является наиболее известной и широко используемой при создании инструментария для оценки сложности программ на различных языках. Цикломатическая сложность вычисляется для графа потока управления процедуры или функции по формуле V(G) = e – n + 2p, где e - количество дуг, n - количество вершин, p - число компонент связности графа потока управления. Число компонент связности рассматривается как количество дуг, которые надо добавить для преобразования графа в сильно связный, т.е. любые две вершины которого взаимно достижимы. Для корректных программ сильно связный граф потока управления получается замыканием дугами вершин, соответствующих точкам выхода, на вершины точек входа. Мера Мак-Кейба может вычисляться для всей системы, если построен общий граф потока управления на основе графа вызовов, либо для отдельных модулей, классов, методов и

других единиц. Для правильной и хорошо структурированной программы с одной точкой входа и одной точкой выхода p=1 (т.к. достаточно замкнуть граф потока управления одной дугой из точки выхода в точку входа). Поэтому формула цикломатической сложности часто встречается в литературе в виде V(G)=e - n + 2; понятно, что в таком виде эта метрика не применима при анализе произвольных программ, а предназначена для оценки программ, разрабатываемых в соответствии с теми или иными требованиями по стилю программирования. Следует отметить, что оценка цикломатической сложности не различает циклические и условные конструкции, а также сложность предикатов (например, выражаемую числом входящих в них переменных). Для устранения недостатков был разработан ряд модификаций метрики Мак-Кейба, в частности, метрики Майерса, Хансена, Пивоварского. Однако простота вычисления исходной метрики Мак-Кейба обуславливает ее широкое распространение и использование как самостоятельно, так и в составе гибридных метрик сложности. Данную метрику можно применять при решении первой и второй задач оценки бинарного кода (классификация приложений и оценка трудоёмкости анализа их частей). Для построения профиля сложности данная метрика непригодна.

2. **Интервальная метрика Майерса** (Myers). В качестве оценки используется интервал [V(G),V(G)+h], где h для простых предикатов равно нулю, а для n-местных h=n-1 (n-местный предикат зависит от n переменных). Данный метод позволяет различать разные по сложности предикаты, однако на практике он почти не применяется. Применительно к анализу бинарного кода, метрика Майерса не имеет заметных преимуществ перед простой цикломатической сложностью при решении любой из трех задач.
3. **Метрика Хансена** (Hanson). Сложность программы (функции) оценивается парой (цикломатическая сложность, число операторов); тем самым повышается чувствительность метрики к структурированности программы. Эта метрика может быть применена для оценки сложности анализа бинарного кода статическими методами, когда аналитику известен размер всего приложения и число инструкций в составляющих его функциях. При анализе трасс метрика Хансена даст различные пары значений в зависимости от того, какие пути в графе потока управления были реализованы в имеющихся трассах. Для построения профиля сложности метрика Хансена непригодна.
4. **Топологическая мера Чена** (Chen). Сложность программы определяется числом пересечений границ между областями,

образуемыми графом программы. Этот метод применим только к структурированным программам с последовательным соединением управляющих конструкций. Для неструктурированных программ мера Чена существенно зависит от условных и безусловных переходов. В этом случае можно указать нижнюю и верхнюю границы меры – 2 и m+1, где m - число логических операторов при их взаимной вложенности. Если граф потока управления имеет только одну компоненту связности, мера Чена совпадает с цикломатической мерой Мак-Кейба.

5. **Метрика Вудворда** (Woodword) определяется как число узлов (точек пересечения) линий передачи управления на полях листинга программы. Таким образом, эта метрика пригодна лишь для статического представления программы. Кроме того, в ней не учитывается ни размер кода, ни сложность предикатов. Глубина вложенности управляющих конструкций также учитывается не всегда аккуратно – так, например, если вложенный условный оператор if не имеет ветви else, то значение меры Вудворда будет на 1 меньше, чем в случае вложенного if с обеими ветвями.
6. **Метрики Харрисона** и **Мейджела** (Harrison & Magel) учитывают уровень вложенности и размер программы. Каждой вершине графа потока управления присваивается *начальная сложность* (например, посредством мер Холстеда). Для каждой предикатной вершины выделяется ее *сфера влияния* – подграф, порожденный вершинами, которые являются концами исходящих из нее дуг, а также вершинами, достижимыми из каждой такой вершины (нижняя граница подграфа), и вершинами, лежащими на путях из предикатной вершины в какую-нибудь нижнюю границу. *Приведенная сложность* предикатной вершины есть сумма начальных или приведенных сложностей вершин, входящих в ее сферу влияния, плюс начальная сложность самой предикатной вершины. *Функциональная мера* (SCOPE) программы - это сумма приведенных сложностей всех вершин управляющего графа. *Функциональное отношение* (SCORT) - это отношение числа вершин в управляющем графе к его функциональной сложности, причем из числа вершин исключаются терминальные. Функциональное отношение может принимать разные значения для графов с одинаковым цикломатическим числом, тем самым повышая чувствительность метрик этой группы к вложенности управляющих конструкций. Метрики Харрисона и Мейджела можно применять для решения первой и второй задач оценки бинарного кода, то есть при классификации приложений по сложности и для оценки

трудоемкости анализа. Профилирование приложений по этим метрикам для поиска механизмов защиты представляется как минимум сомнительным.

7. **Метрика Пивоварского** позволяет учесть различия не только между последовательными и вложенными управляющими конструкциями, но и между структурированными и неструктурированными программами. Она выражается отношением N(G) = v*(G) + СУММАРi , где v*(G) - модифицированная цикломатическая сложность, при вычислении которой оператор CASE с n выходами рассматривается как один логический оператор, а не как n - 1 операторов. Pi - глубина вложенности i-й предикатной вершины, то есть число всех сфер влияния предикатов, которые либо полностью содержатся в сфере рассматриваемой вершины, либо пересекаются с ней. Глубина вложенности увеличивается за счет вложенности не самих предикатов, а сфер влияния. Таким образом, мера Пивоварского возрастает при переходе от последовательных программ к вложенным и далее к неструктурированным, что является ее преимуществом перед многими другими мерами данной группы. Применимость метрики Пивоварского к задачам анализа бинарного кода аналогична применимости предыдущей группы метрик (Харрисона и Мейджела).
8. **Метрика граничных значений** (boundary value) вычисляется как S0=1-(v-1)/Sa
   где S0 - относительная граничная сложность программы,

   Sa - абсолютная граничная сложность программы,
   v - общее число вершин графа потока управления.

   Абсолютная граничная сложность определяется как сумма приведенных сложностей всех вершин графа. В свою очередь, приведенная сложность вершины определяется равной 1 для принимающих вершин, кроме конечной, для которой приведенная сложность равна 0. Принимающая вершина – такая вершина графа потока управления, положительная степень которой (т.е. число исходящих дуг) не превышает 1. Если положительная степень больше или равна 2, то вершина является вершиной отбора. Приведенная сложность вершины отбора определяется числом вершин минимального подграфа, для которого эта вершина отбора является начальной, а нижней границей является вершина, в которую можно попасть из любой другой вершины подграфа. Например, для ветвления if-then-else без вложенности подграф состоит из 4 вершин (добавляется следующий за оператором ветвления базовый блок), приведенная сложность вершины отбора равна 3, приведенные

сложности трех других вершин равны 1 (если вершина, являющаяся нижней границей этого подграфа, не является конечной в полном графе потока управления; в последнем случае ее сложность – 0).

Применимость метрики граничных значений к задачам анализа бинарного кода аналогична применимости двух предшествующих метрик. Некоторые дополнительные возможности данной метрики связаны с тем, что она различным образом оценивает реализующие одну и ту же функциональность последовательности операторов ветвлений и переключатель CASE (для которого значение меры граничных значений существенно ниже). Если одновременно учитывать другие метрики (например, цикломатическую сложность), то по меньшему значению относительной сложности можно различать программные единицы с большим переключателем CASE, и программные единицы с большим числом ветвлений (цикломатическая сложность в обоих случаях будет примерно одинакова).

9. **Метрика Шнейдевинда** (Schneidewind) выражается через число возможных путей в графе потока управления. На практике эта метрика применяется редко и в основном при оценке тестовых покрытий. В большинстве обзоров метрик программного кода эта метрика подробно не рассматривается, а лишь упоминается. Представляется, что она не имеет преимуществ перед рассмотренными выше метриками применительно к анализу двоичного кода приложений.

Из перечисленных метрик при анализе программ в двоичных кодах могут быть рассмотрены меры Мак-Кейба, Хансена, Харрисона и Мейджела, а также метрика Пивоварского и метрика граничных значений. При этом, при вычислении метрик Харрисона, Мейджела, Пивоварского и граничных значений вычисляются промежуточные значения (приведенные сложности и глубины вложенности), которые могут быть учтены при построении комбинированных и гибридных метрик для формирования профиля сложности. Метрика Майерса требует дополнительного анализа каждого предиката для определения числа переменных, от которых он зависит. Мера Чена применима только для структурированных программ. Метрики Вудворда и Шнейдевинда не имеют преимуществ перед другими перечисленными метриками, а при динамическом анализе они не учитывают сложность не выполнявшихся ветвей программы; впрочем, то же самое справедливо и для остальных метрик.

Применительно к построению профиля сложности трасс представляется, что ни одна из рассмотренных метрик не является для этого достаточной. Необходимы гибридные метрики, сочетающие меры сложности потока управления с количественными мерами. Как вариант,

такая метрика может быть построена на основе меры Пивоварского либо метрики граничных значений, и относительной меры Джилба или же размера базового блока.

2.3. Метрики сложности потока данных программы

В отличие от метрик сложности управления, метрики сложности потока данных в большей степени ориентированы на оценку исходного текста программ на языках высокого уровня, соответственно, с высокоуровневыми типами данных. Модификация большинства приведенных ниже метрик для низкоуровневого представления, то есть для бинарного кода, как минимум требует некоторого анализа бинарного кода для восстановления данных. Поэтому наиболее сложные метрики ниже лишь упоминаются, но подробно не рассматриваются.

1. **Метрика Чепина** (Chepin) оценивает информационную прочность отдельно взятого модуля по характеру использования переменных из списка ввода-вывода. Всё множество переменных разбивается на 4 группы: P (вводимые для расчетов и для обеспечения вывода), M (модифицируемые, или создаваемые внутри программы), C (управляющие), T (паразитные, то есть не используемые). Переменные, выполняющие несколько функций, учитываются в каждой функциональной группе. Метрика Чепина выражается формулой Q = a1*P + a2*M + a3*C + a4*T, где a1, a2, a3, a4 - весовые коэффициенты. По мнению автора, эти коэффициенты следует принять равными 1, 2, 3 и 0.5 соответственно, то есть учитывается, что переменные группы C влияют также на поток управления программы. Паразитные переменные не увеличивают сложность потока данных, но затрудняют понимание программы.
2. **Метрика спена** основывается на локализации обращений к данным внутри каждой программной единицы. Спен (span) - это число обращений к переменной между её первым и последним появлением в программе (переменная, встретившаяся n раз, имеет спен, равный n-1).
3. **Метрика обращений к глобальным переменным**. В зависимости от наличия в программе реальных обращений к переменной, формируются фактические и возможные пары (модуль, глобальная переменная), говорящие о том, сколько раз модуль действительно получал доступ к глобальным переменным, и сколько раз он мог этот доступ получить. Отношение фактического числа обращений к возможному показывает вероятность несанкционированного изменения глобальной переменной.

4. **Метрика Кафура** (Henry & Kafura) строится на основе концепции информационных потоков (встречается также под названием “Fan in/out complexity”). Эта мера оперирует локальными и глобальными потоками информации и вводит оценку информационной сложности процедуры и модуля (через потоки либо относительно некоторой структуры данных). Информационная сложность учитывает также сложность текста, измеряемую через одну из метрик объема, например, Холстеда или LOC; может учитываться также цикломатическая сложность.
5. **Информационная сложность** модуля относительно структуры данных вычисляется на основе числа процедур, которые только обновляют, только читают, либо и обновляют, и читают некоторую структуру данных.
6. **Мера Овиедо** (Oviedo). Программа разбивается на линейные непересекающиеся участки – так называемые лучи операторов, которые образуют граф потока управления программы. Мера сложности каждого луча определяется, как сумма количеств определяющих вхождений для каждой используемой в луче переменной. Автор метрики предполагает, что найти отношение между определениями и использованиями переменной внутри луча проще, чем между лучами, и что число различных определяющих вхождений в каждом луче важнее, чем общее число использующих вхождений переменной в каждом луче. Встречается также иное определение меры Овиедо: $C = aCF + bDF$,
   где CF – сложность потока управления, учитывающая только число дуг графа;
   DF – сложность потока данных, вычисляемая как сумма сложностей потоков данных базовых блоков программы, причем сложность потока данных блока есть число переменных, которые используются, но не определяются в блоке;
   a, b – весовые коэффициенты, которые могут быть приняты равными 1.

В чистом виде все перечисленные метрики недостаточны для того, чтобы охарактеризовать сложность анализа трасс программ в бинарных кодах. Применение метрик сложности потока данных к двоичному коду осложняется также тем, что компилятор может использовать для размещения переменных как память, так и регистры процессора, и требуется дополнительный анализ с целью сведения переменных в единый список.

Метрика Чепина может представлять интерес главным образом при создании комбинированных и гибридных метрик сложности запутанного кода, в особенности, использующего непрозрачные переменные (подробнее об этом – в разделах 3 и 4). Метрика спена легко

вычислима и применима к трассам программ. Метрики обращений к глобальным переменным и Кафура представляются слишком сложными для использования при анализе трасс. Метрика сложности модуля относительно структуры данных не применима для двоичных кодов, так как информация о сложных структурах данных потеряна при компиляции. Мера Овиедо может рассматриваться, как переходная от «чистых» метрик сложности потока данных к комбинированным, так как учитывает поток управления и возрастает с числом «межблочных ссылок», когда переменная определяется и используется в разных базовых блоках программы. К сожалению, о практическом применении этой метрики в реальных проектах информация отсутствует.

2.4. Комбинированные метрики сложности управления и данных

Метрики этого класса близки как к количественным метрикам, метрикам сложности потока управления, так и к метрикам сложности потока данных. Они определяют сложность программы как на основе количественных подсчетов, так и на основе анализа управляющих структур. В основном метрики данного класса применяются для оценки сложности программ при проектировании и реализации на языках высокого уровня, поэтому здесь их можно было бы и не приводить. Однако можно просто перечислить ряд метрик для полноты классификации:

1. **Тестирующая М-мера** возрастает с глубиной вложенности и учитывает протяженность программы.
2. **Метрика на основе регулярных выражений** вычисляется на основе подсчета суммарного числа символов (операндов, операторов, скобок) в регулярном выражении, описывающем граф потока управления программы.
3. **Метрика связанности модулей** (modules’ cohesion) программы. Различают связанность по данным, по структуре данных, по управлению, по общей области (по глобальным данным), по содержимому, а также внешнюю связанность, связанность при помощи сообщений, подклассовую связанность, связанность по времени и отсутствие связанности.
4. **Мера Колофелло** (Kolofello) используется при оценке стабильности модуля.
5. **Метрика Мак-Клура** (McCloore) ориентирована на хорошо структурированные иерархически организованные программы с одной точкой входа и одной точкой выхода в каждом модуле, выполняющем ровно одну функцию.
6. **Метрика Берлингера** (Berlinger) основана на информационной концепции и вычисляется с учетом частоты и вероятности появления символов.

Как уже упоминалось выше, метрики этой группы не представляют интереса при оценке сложности бинарного кода, в частности, динамическими методами анализа.

2.5. Объектно-ориентированные метрики и метрики надежности

Наборы метрик **Мартина** (Martin)**, Чидамбера и Кемерера** (Chidamber & Kemerer) используются при оценке программ на объектно-ориентированных языках программирования. В контексте задачи анализа приложений в бинарных кодах эту группу метрик можно не рассматривать.

Аналогично, не рассматриваются применительно к анализу бинарного кода и метрики надежности программного обеспечения, близкие к количественным, но основанные на количестве ошибок и дефектов в программе. Примеры таких метрик: количество структурных изменений, произведенных с момента прошлой проверки; количество ошибок, выявленных при просмотре кода; количество ошибок, выявленных при тестировании; количество необходимых структурных изменений, необходимых для корректной работы программы.

2.6. Гибридные метрики

Метрики данного класса основываются на более простых метриках и представляют собой их взвешенную сумму:

1. **Метрика Кокола** (Cocol) определяется как
   $H\_M = (M + R_1 * M(M_1) + ... + R_n * M(M_n)/(1 + R_1 + ... + R_n)$, где

   M - базовая метрика, M_i - другие интересные элементарные меры, R_i - некоторые корректно подобранные коэффициенты, $M(M_i)$ - функции. Функции и коэффициенты вычисляются с помощью регрессионного анализа или анализа задачи для конкретной программы. Видимо, при этом должны также учитываться эмпирические данные о взаимосвязи используемых элементарных мер в предыдущих проектах, либо на предыдущих этапах жизненного цикла данного оцениваемого проекта. Автор метрики выделил три модели для мер: Мак-Кейба, Холстеда и SLOC, причем в качестве базовой используется метрика Холстеда. Эти модели получили название "наилучшая", "случайная" и "линейная".
2. **Метрика Зольновского, Симмонса, Тейера** (Zolnovskiy, Simmons & Tayer) также представляет собой взвешенную сумму различных индикаторов. Существующие варианты данной метрики учитывают высокоуровневые характеристики программ – структуру, взаимодействие, объем, данные либо

сложность интерфейса, вычислительную сложность, сложность ввода/вывода, читабельность.

В исходном виде ни метрика Кокола, ни метрики Зольновского, Симмонса, Тейера не представляют ценности при анализе бинарного кода. Однако может представлять интерес подбор иных мер из числа рассмотренных выше для формулы Кокола применительно к анализу приложений в бинарных кодах. Подбор метрик, функций и коэффициентов применительно к анализу приложений в бинарных кодах должен быть предметом отдельного исследования.

3. Запутывающие преобразования и их влияние на известные метрики

Различаются 4 основные группы запутывающих преобразований - запутывание форматирования исходного текста (layout obfuscation), запутывание данных (data obfuscation), запутывание управления (control obfuscation) и превентивные трансформации (preventive transformation). Рассмотрим запутывающие преобразования, входящие в эти группы, с точки зрения их влияния на известные метрики кода.

3.1. Запутывание форматирования

Данная группа запутывающих преобразований применяется к исходному тексту программы (как правило, на языке высокого уровня). Преобразования этого класса - изменение имен, или "перемешивание" (scramble) идентификаторов, изменение (удаление) форматирования, удаление комментариев, удаление отладочной информации. Лишь последнее преобразование (удаление отладочной информации) выполняется на уровне объектной программы.

Перечисленные преобразования не только являются тривиальными с точки зрения их реализации и преодоления, но и не представляют интереса в контексте исследования метрических характеристик двоичного кода. Действительно, они оказывают влияние лишь на значения таких мер, как SLOC, среднее количество строк, количество пустых строк, количество или процент комментариев, оценка стилистики. Сложность анализа бинарного кода приложения эти преобразования, за исключением удаления отладочной информации, не изменяют. Удаление же отладочной информации либо не влияет на известные метрики (в первую очередь, на метрики сложности потока управления), либо даже незначительно уменьшает меру (применительно к некоторым метрикам сложности данных).

3.2. Запутывание данных

Преобразования запутывания данных меняют размещение в памяти, кодирование, агрегацию (состав) или порядок элементов в объектах данных. Рассмотрим эти преобразования по группам:

1. **Преобразования размещения и кодирования данных**. Замены локальных переменных глобальными и наоборот, продвижение переменных, изменение времени жизни переменных, разбиение переменных, замена статических данных процедурными увеличивают меры сложности данных (метрика Чепина, метрика обращений к глобальным переменным, метрика спена, метрика Кафура). Введение функции кодирования (например, замена простой целочисленной переменной на выражение из переменных и констант), помимо метрик сложности данных, также может учитываться интервальной мерой Майерса сложности потока управления, но только в случае, если функция кодирования увеличивает сложность предикатов. Замена статических данных процедурными также может увеличивать различные меры сложности потока управления, поскольку при этом преобразовании генерируется функция вычисления статических данных и меняется граф потока управления, что может привести к росту мер Мак-Кейба, Майерса и Хансена; перечисленные метрики увеличиваются в большей степени, если генерируемая функция не вызывается, а разбивается на компоненты, непосредственно встраиваемые в поток управления запутываемых программных единиц. Значение гибридной меры Кокола также может незначительно возрасти в зависимости от весового коэффициента при мере Мак-Кейба.
2. **Преобразования агрегации данных**. В группу преобразований агрегации входят слияние скалярных переменных, трансформация массивов (разбиение, слияние, складывание, выравнивание), изменение отношений наследования классов и т.д. В двоичном коде часть этих преобразований (как, например, изменение отношений наследования классов) просто теряется при компиляции, а на другие преобразования (преобразования массивов) могут накладываться способы отображения структур данных и методы оптимизации, используемые компиляторами. Преобразования массивов могут не только увеличивать, но и уменьшать метрики сложности структур данных. Следует учесть, что изменения меры сложности данных при запутывании агрегации данных возможно оценить, скорее, на уровне исходного кода приложения; применить эту меру для трасс не представляется возможным. Еще менее целесообразно использование метрики сложности структур данных для профилирования сложности трасс.

3. **Преобразования упорядочивания данных**. Аналогично преобразованиям агрегации, учёт этой группы запутывающих преобразований при построении метрики сложности данных на уровне двоичного кода (тем более, профиля сложности трассы) не представляется возможным и целесообразным. Возможно косвенное влияние переупорядочивания данных на метрики сложности управления, так как в код добавляется непрозрачная функция отображения; это может несколько увеличить меры Мак-Кейба, Майерса, Хансена и Кокола.

Итого, в первом приближении для учета запутывания данных представляется целесообразным использовать меры Мак-Кейба, Чепина, Кафура. Метрики Майерса, Хансена и мера спена менее удобны, так как только на их основе нельзя построить профиль сложности трассы; однако они могут быть учтены в формулах комбинированных метрик, таких, например, как мера Кокола.

3.3. Запутывание потока управления

Запутывание потока управления, в отличие от запутывания данных, способно очень сильно увеличить сложность программ в двоичных кодах. Соответственно, учитывающие эти преобразования меры сложности являются более актуальными в контексте рассматриваемой задачи как по отдельности, так и при разработке гибридных метрик.

1. **Преобразования агрегации**. *Вставка функций* (inline) увеличивает почти все меры сложности потока управления, основанные на оценке цикломатической сложности (если, конечно, вставляемая функция достаточно сложна и не состоит из одного базового блока). Наоборот, *вынос функции* (outline) уменьшает эти меры, следовательно, не учитывается адекватно ни одной из метрик сложности потока управления; однако автоматический поиск тривиальных outline-функций возможен посредством вычисления количественной ABC-меры, которая в таких случаях зачастую равна 1 или даже 0. Это преобразование можно учесть также через общее число функций в формуле комбинированной метрики. *Переплетение функций* уменьшает их общее количество, но увеличивает как метрики размера и сложности управления, так и сложность данных (за счет объединения списков параметров). Следует дополнительно заметить, что переплетение функций может возникать и в качестве побочного эффекта *замены инструкции возврата* из функции инструкцией перехода – тогда точка выхода из функции не всегда может быть распознана корректно, и тогда часть кода вызывающей функции ошибочно относится к вызываемой. *Клонирование кода* увеличивает цикломатическую сложность и метрики Джилба, а также другие количественные

меры. *Раскрутка циклов* увеличивает количественные метрики (размер кода), меру Хансена и меру спена (сложность данных возрастает за счет большего числа обращений к массиву в теле цикла). *Переупорядочивание операторов, выражений и инструкций* на метрики сложности потока управления влияет непредсказуемым образом.

2. **Преобразования вычислений.** *Вставка мёртвого и избыточного кода* увеличивает количественные метрики и может усложнять поток управления (все соответствующие метрики). То же относится и к другим *вычислительным преобразованиям* (преобразование графа потока управления к несводимому, диспетчер, расширение условий циклов и т.д.). *Диспетчеризация* очень эффективно искажает поток управления, но приводит к увеличению приведенной сложности вершин графа потока управления, что позволяет эффективно пользоваться метрикой граничных значений. Отдельно следует рассмотреть механизм непрозрачных предикатов, на использовании которого основано значительное число преобразований вычислений.
3. **Непрозрачные предикаты** увеличивают цикломатическую сложность программы и все меры, использующие эту величину (Мак-Кейба, Майерса, Хансена), а также меры Чена, Вудворда, Харрисона и Мейджела, Пивоварского, Шнейдевинда и меру граничных значений. Увеличивается также количественная мера Джилба, в особенности, в случае учета уровня вложенности условных конструкций. Непрозрачные предикаты, основанные на добавленных *непрозрачных переменных*, увеличивают также сложность данных (метрика Чепина).
4. **Табличная интерпретация** (программная виртуализация) является одним из наиболее эффективных, но затратных способов запутывания потока управления. Оценка этого способа запутывания с помощью известных мер представляется затруднительной, так как зависит от сочетания различных факторов. Добавление в программу интерпретатора и трансляция фрагментов кода в его систему команд увеличивает трудоёмкость анализа приложения в целом, но при этом может уменьшать ряд метрик сложности – например, виртуализация части кода большой и сложной функции уменьшает ее количественные метрики и меры сложности потока управления, добавляя лишь один массив «данных». На практике, однако, можно исходить из того, что данный метод запутывания является не только эффективным, но и очень затратным. Как следствие, интерпретируется по возможности небольшой участок кода, а сам интерпретатор достаточно велик, и его наличие будет отражено в количественных метриках

приложения. В то же время, табличный интрепретатор имеет характерный граф потока управления с большим числом принимающих вершин при одной вершине отбора (большой переключатель CASE), что позволяет рассчитывать на успешное применение для его поиска метрики граничных значений.

Итак, преобразования запутывания потока управления могут оцениваться количественными метриками – ABC и Джилба, мерами на основе цикломатической сложности (Мак-Кейба, Майерса, Хансена), а также мерой граничных значений. Для учёта преобразований выноса функций возможно использование ABC-метрики, либо учет в формуле гибридной метрики общего числа функций. Однако вставка и переплетение функций уменьшают эту общее число функций приложения, поэтому целесообразность использования такой меры сомнительна, либо она должна учитываться в гибридной метрике с незначительным весовым коэффициентом.

3.4. Превентивные трансформации

Существующие деобфускаторы применяют для поиска непрозрачных предикатов достаточно небольшой набор известных технических приемов; как следствие, при запутывании программа может быть дополнительно модифицирована таким образом, чтобы затруднить автоматическое распознавание непрозрачных конструкций. Отладчики, декомпиляторы и деобфускаторы также обычно не свободны от ошибок и уязвимостей, которые могут быть целью запутывающих преобразований. Такие преобразования, называемые превентивными, могут быть двух типов – врождённые (затрудняющие применение известных методов автоматического распутывания) и целевые (использующие известные проблемы существующих деобфускаторов и декомпиляторов). При этом само преобразование может быть очень простым – например, добавление произвольной мусорной инструкции в определенном месте кода приводит к отказу декомпилятора. Обнаружение подобных целевых преобразований на основе метрик сложности кода в настоящий момент не представляется возможным. Врожденные преобразования могут незначительно увеличивать известные меры, например, за счёт введения фиктивных зависимостей по данным.

4. Заключение

Необходимо отметить, что ни одной универсальной метрики среди известных мер сложности кода не существует. Любые рассмотренные метрические характеристики анализируемых программ должны либо использоваться совместно и в зависимости друг от друга, либо

применяться в зависимости от конкретной задачи. Возможно применение гибридных метрик, однако они зависят от более простых, базовых метрик и, следовательно, не могут рассматриваться в качестве универсальных. Применительно к анализу сложности запутанного кода, получен такой список наиболее подходящих базовых метрик:

1. Количественные метрики: среднее число инструкций в функции, относительная сложность по Джилбу, ABC-метрика и обратная метрика размера базовых блоков; три последние метрики пригодны не только для классификации приложений и оценки трудоемкости анализа, но и для построения профиля сложности.
2. Метрики сложности потока управления: цикломатическая сложность по Мак-Кейбу, метрика Хансена, метрика Харрисона и Мейджела, метрика Пивоварского, метрика граничных значений. Эти метрики могут быть применены для решения первых двух задач – классификации приложений по сложности анализа и оценки трудозатрат на анализ частей одного приложения. Кроме того, метрика граничных значений в совокупности с иными метриками может использоваться для поиска таких видов запутывания, как диспетчер и табличная интерпретация.
3. Метрики сложности потока данных: метрика Чепина, спена, Кафура. Эти метрики могут быть включены в формулы гибридных метрик, но требуется дополнительное исследование для определения их применимости в разных целях, а также для вычисления коэффициентов при значениях данных мер.
4. Гибридная метрика Кокола (где базовая и другие элементарные метрики должны подбираться в зависимости от задачи) может применяться для – классификации приложений по сложности анализа и для оценки трудозатрат при анализе одного приложения.

Дальнейшие исследования метрик сложности запутанных приложений должны быть направлены на проведение серии экспериментов с вычислением простых метрик, формированием шкал сложности и трудоемкости, а также построением профилей анализируемых приложений и трасс на основе пригодных для этого метрик. Будет реализован плагин системы TrEx для экспериментов с различными известными метриками, а также с разрабатываемыми метриками запутанности кода. Результатом таких экспериментов будет уточнение набора базовых элементарных метрик и весовых коэффициентов для гибридной метрики запутанности кода, позволяющей автоматизировать поиск защитных механизмов в бинарном коде приложения.

Список использованных источников

[1] T.J. McCabe, "A complexity measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, December, 1976

[2] Arthur H.Watson, Thomas J. McCabe, “Structured Testing: A Testing Methodology Using Cyclomatic Complexity Metric”, NIST Special Publication 500-235, 1996 (http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/title.htm)

[3] Myers, G., “An Extension to the Cyclomatic Measure of Program Complexity”, SIGPLAN Notices, October 1977

[4] Hanson, D., “Printing Common Words”, Communications of the ACM, July 1987

[5] Henry, S. and D.Kafura and K.Harris, “On the Relationship Among Three Software Metrics”, 1981 ACM Workshop|Symposium on Measurement and Evaluation of Software Quality, March 1981

[6] Kafura, D. and G.Reddy, “The Use of Software Complexity Metrics in Software Maintenance”, IEEE Transactions on Software Engineering, March 1987

[7] Christian Collberg, Clark Thomborson, Douglas Low, A Taxonomy of Obfuscating Transformations, Technical Report #148, Departament of Computer Science, The University of Auckland, Auckland, New Zealand.

[8] Чернов А.В., Анализ запутывающих преобразований программ, Труды Института Системного программирования РАН, 2003

[9] Matias Madou, Bertrand Anckaert, Bruno De Bus, Koen De Bosschere, Jan Cappaert, Bart Preneel, On the Effectiveness of Source Code Transformations for Binary Obfuscation, 2008

[10] Богданов Д.В., "Стандартизация жизненного цикла программных средств", СПб – 2000

[11] Hassan Raza Bhatti, “Automatic Measurement of Source Code Complexity”, Master’s Thesis, Lulea University of Technology, Lulea, Sweden, 2011

[12] Милютин А., «Метрики кода программного обеспечения» http://www.viva64.com/ru/a/0045/

[13] Новичков А., «Метрики кода и их практическая реализация в IBM Rational ClearCase» http://www.viva64.com/go.php?url=241