­­­­Белорусский государственный университет информатики и радиоэлектроники

Кафедра программного обеспечения информационных технологий

­

«Практическая реализация метрик кода в Visual Studio»

­

Выполнили:

ст. гр. 820601

Шведов А.Р.

Судаков Б.Д.

Минск 2019

**Введение**

Статический анализ кода – анализ исходных кодов приложения, производимый (в отличие от динамического анализа) без реального выполнения проверяемых программ. В подавляющем большинстве случаев статический анализ осуществляется над какой-либо версией исходного кода, обычно последней.

Одним из главных преимуществ статического анализа является возможность существенного снижения стоимости устранения дефектов в программе. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления. Так, согласно данным, исправление ошибки на этапе тестирования обойдется до 10 раз дороже, чем на этапе конструирования программы [1].

Задачи, решаемые программами статического анализа кода можно разделить на 3 категории:

1. Выявление ошибок в программах.
2. Рекомендации по оформлению кода. Некоторые статические анализаторы позволяют проверять, соответствует ли исходный код, принятому в компании стандарту оформления кода. Имеется в виду контроль количества отступов в различных конструкциях, использование пробелов/символов табуляции и так далее.
3. Подсчет метрик. Метрика программного обеспечения - это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций. Существует большое количество разнообразных метрик, которые можно подсчитать, используя те ли иные инструменты.

**Значение метрик кода и программные измерения в Visual studio**

Разработчики могут использовать Visual Studio для создания данных метрик кода, которые измеряют сложность и удобство обслуживания управляемого кода. Данные метрик кода могут создаваться для всего решения или отдельного проекта.

В следующем списке показаны результаты метрик кода, которые вычисляет Visual Studio:

* **Индекс удобства обслуживания (Maintainability Index)** —расчитывается по формуле *MI = MAX(0, (171 — 5.2 \* ln(HV) — 0.23 \* CC — 16.2 \* ln(LoC)) \* 100 / 171),* где HV - Halstead Volume, вычислительная сложность. Чем больше операторов, тем больше значение этой метрики, CC – Cyclomatic Complexity, эта метрика описана ниже, LoC – количество строк кода. Метрика вычисляет значение индекса от 0 до 100, представляющее относительную простоту обслуживания кода. Высокое значение означает лучшую сопровождаемость. Для быстрого обнаружения проблем в коде можно использовать цветовую маркировку. Зеленая Оценка находится в диапазоне от 20 до 100 и указывает на то, что код обладает хорошей сопровождаемостью. Желтая Оценка находится в диапазоне от 10 до 19 и указывает, что код является умеренно поддерживаемым. Красная Оценка — это оценка между 0 и 9 и указывает на низкую сопровождаемость.
* **Сложность организации циклов (Cyclomatic complexity)** — измеряет структурную сложность кода. Он создается путем вычисления количества различных путей кода в потоке программы. Программа, имеющая сложный поток управления, требует больше тестов для достижения хорошего объема протестированного кода и менее сопровождаемой.
* **Глубина наследования (Depth of Inheritance)** — указывает количество различных классов, которые наследуют друг от друга, вплоть до базового класса. Глубина наследования аналогична взаимосвязанности классов в том, что изменение базового класса может повлиять на любые из его унаследованных классов. Чем выше это число, тем более глубокое наследование и тем выше вероятность внесения изменений в базовый класс, что приводит к критическому изменению. Для более глубокого наследования низкое значение хорошо, а высокое значение является недопустимым.
* **Взаимосвязь** **классов (Class Coupling)** — измеряет связь с уникальными классами через параметры, локальные переменные, возвращаемые типы, вызовы методов, универсальные экземпляры или шаблоны шаблонов, базовые классы, реализации интерфейса, поля, определенные на внешних типах, и атрибут. параметризацию. Хорошая разработка программного обеспечения определяет, что типы и методы должны иметь высокую связность и низкую связь. Высокая связь указывает на проект, который трудно использовать и поддерживать из-за множества взаимозависимостей от других типов.
* **Строки кода (Lines of Code)** — указывает приблизительное количество строк в коде. Количество основывается на коде IL (управляемый код) и, следовательно, не является точным числом строк в файле исходного кода. Большое число может означать, что тип или метод пытается выполнить слишком много усилий и должны быть разделены. Оно также может указывать на то, что тип или метод может быть трудно поддерживать.

# **Преимущества**

Инструменты статического анализа позволяют выявить большое количество ошибок этапа конструирования, что существенно снижает стоимость разработки всего проекта. Так же к плюсам можно отнести:

1. **Полное покрытие кода.** Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удается протестировать другими методами. Это позволяет находить дефекты в обработчиках редких ситуаций, в обработчиках ошибок или в системе логирования.
2. Статический анализ **не зависит от используемого компилятора и среды**, в которой будет выполняться скомпилированная программа. Это позволяет находить скрытые ошибки, которые могут проявить себя только через несколько лет. Например, это ошибки неопределенного поведения. Такие ошибки могут проявить себя при смене версии компилятора или при использовании других ключей для оптимизации кода.
3. **Быстрое и легкое обнаружение опечаток и синтаксических ошибок.** Как правило, нахождение этих ошибок другими способами является кране неэффективной тратой времени и усилий. Обидно после часа отладки обнаружить, что ошибка заключается в выражении вида "strcmp(A, A)". Обсуждая типовые ошибки, про такие ляпы, как правило, не вспоминают. Но на практике на их выявление тратится существенное время.

# **Критика**

1. **Нет диагностики утечек памяти и параллельных ошибок.** Чтобы выявлять подобные ошибки, фактически необходимо виртуально выполнить часть программы. Это крайне сложно реализовать. Также подобные алгоритмы требуют очень много памяти и процессорного времени. Как правило, статические анализаторы ограничиваются диагностикой простых случаев. Более эффективным способом выявления утечек памяти и параллельных ошибок является использование инструментов динамического анализа.
2. **Ложно-позитивные срабатывания.** Понять, указывает анализатор на ошибку или выдал ложное срабатывание, может только программист. Необходимость просматривать ложные срабатывания отнимает рабочее время и ослабляет внимание к тем участкам кода, где в действительности содержатся ошибки.
3. **Человеческий фактор.** Метрики бывают искаженными, ведь программист может стремится написать эффективное и правильное решение, а не оптимизировать показатели метрик.

## **Литература**

1. Chess B*. Secure Programming with Static Analysis*. Boston: Addison-Wesley Professional, 2007.
2. Романов В.Ю. *Анализ Метрик для проектирования архитерктуры ПО.* International Journal of Open Information Technologies ISSN: 2307-8162 том 2, номер 3, 2014.
3. Интернет-ресурс: <https://docs.microsoft.com/ru-ru/visualstudio/code-quality/>
4. Интернет-ресурс: <https://github.com/dotnet/roslyn-analyzers>
5. Интернет-ресурс: <https://www.viva64.com/>