Анализ программного кода методами машинного обучения

Научный руководитель: Брыксин Т. А.
Работа Свидченко Олега

Введение

Автоматизация анализа кода

- Анализ кода и рефакторинг - важная часть разработки ПО
- Автоматизация анализа кода ускоряет рефакторинг

Оценка качества кода

- Важно уметь объективно оценивать качество кода
- Формально описать “хороший код” сложно

Цель

- Упростить и/или автоматизировать оценку качества кода
- Научиться предлагать способы улучшения качества кода на основе анализа
- В итоге хочется получить плагин/набор плагинов для IDEA, помогающий программистам в процессе рефакторинга кода

Подзадачи

- Научиться использовать готовые инструменты для численной оценки кода
  (плагин MetricsReloaded)
- Изучить уже существующие статьи по анализу кода с использованием машинного обучения
- Повторить результаты статей, реализовав их на базе плагина
- Попробовать улучшить результаты, добавив свои идеи
- Получить готовый работающий плагин

О MetricsReloaded

- Позволяет рекурсивно считать метрики на синтаксическом дереве разбора программы
- Для работы с Java использует подсистему Program Structure Interface из IDEA SDK
- Имеет множество встроенных метрик
- Позволяет несложно реализовать последующую обработку метрик
- https://github.com/BasLeijdekkers/MetricsReloaded

О MetricsReloaded

В ходе разработки реализованы следующие метрики:

> Unique Operators, Unique Operands, Total Operands, Total Operators, Blanc LoC, Condition Count, Branch Count, Decision Count, Cyclomatic Complexity, Halstead Level, Design Density, Formal Parameters, Distance in Inheritance Tree, Number Of Children, Fan-In, Fan-Out

Также код дополнен ProjectContainerUtil (иногда для подсчета метрики необходимы данные обо всем проекте)

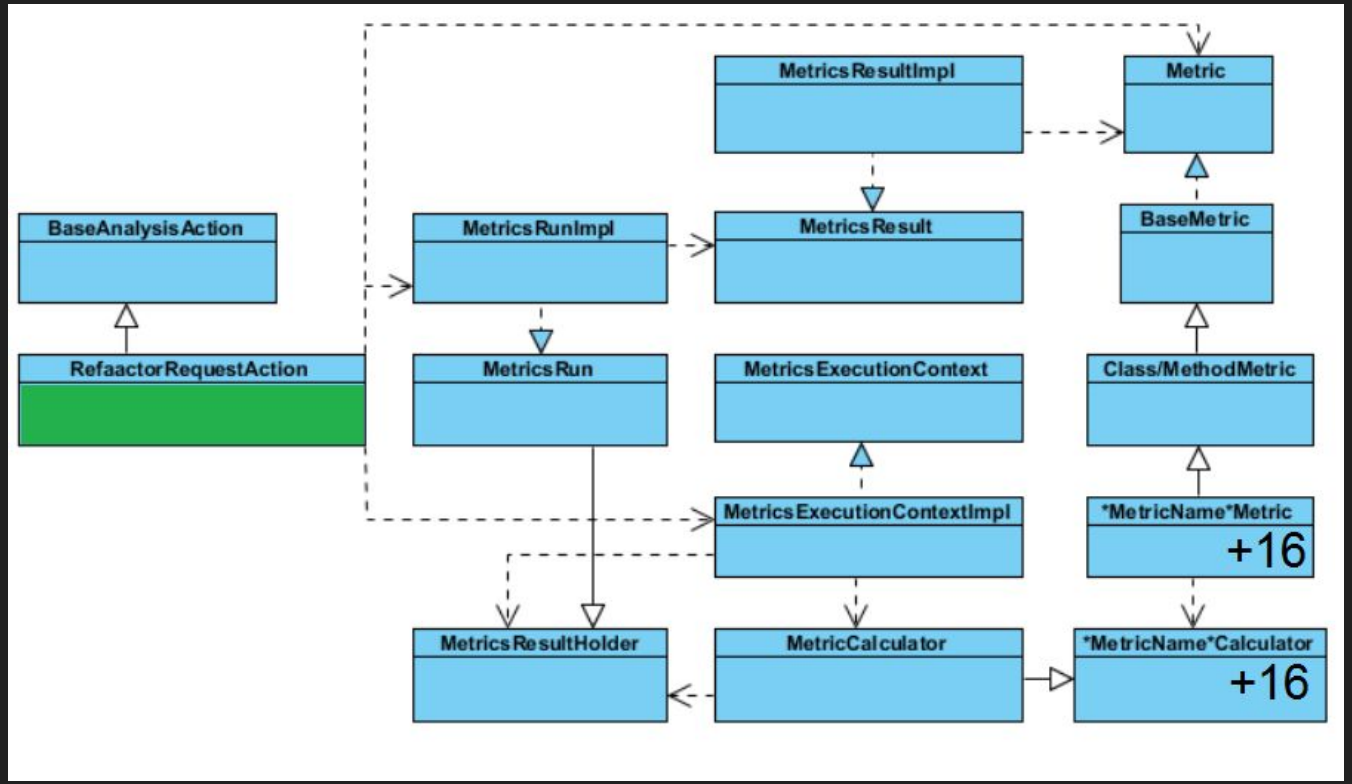
Алгоритм

- Статья Using Software Metrics for Automatic Software Design Improvement от Января 2012 года за авторством Zsuzsanna Marien, Gabriela Czibula, Istvan Gergerly Czibula.
- Идея заключается в перераспределении методов между классами
- За основу взяты четыре метрики: DIT, NOC, Fan-IN, Fan-OUT
- Также оценивается текущая “близость” методов и классов за счет оценки числа общих “интересных” элементов в синтаксическом дереве

Алгоритм

- Функция расстояния определяется через метрики
- Алгоритм: для каждого метода ищем ближайший к нему класс, если такого нет, то добавляем к новому классу.
- Создается список действий, предложенных программисту, состоящий из: Move Method, Create New Class, Remove Old Class

О реализации



Результаты запуска

Искусственный пример, когда выгодно переместить метод (mB1) в другой класс.

```
class A {

    static int a1;
    static int a2;

    static void mA1() {
        a1 = 0;
        mA2();
    }

    static void mA2() {
        a1 = 0;
        a2 = 0;
    }

    static void mA3() {
        a1 = 0;
        a2 = 0;
        mA1();
        mA2();
    }
}
```

```
class B {

    int b1;
    int b2;

    void mB1() {
        A.a1 = 0;
        A.a2 = 0;
        A.mA1();
    }

    void mB2() {
        b1 = 0;
        b2 = 0;
    }

    void mB3() {
        b1 = 0;
        mB1();
        mB2();
    }
}
```

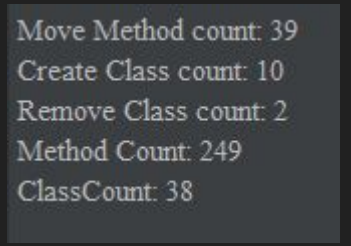
Результаты запуска

- Расстояние от mB1 до B примерно равно 0.44
- Расстояние от mB1 до A примерно равно 0.4
- Значит, метод стоит перенести

	test.A.mA3()	test.A.mA1()	test.B.mB2()	test.B.mB3()	test.A.mA2()	test.A	test.B.mB1()	test.B
test.A.mA3()	0.0	0.2916666666666667	Inf	Inf	0.2886751345948129	0.041666666666666664	0.39747466725706065	Inf
test.A.mA1()	0.2916666666666667	0.0	Inf	Inf	0.3189609868167439	0.2886751345948129	0.42666573660612916	Inf
test.B.mB2()	Inf	Inf	0.0	0.3535533905932738	Inf	Inf	0.46955948623269356	0.2916666666666667
test.B.mB3()	Inf	Inf	0.3535533905932738	0.0	Inf	Inf	0.43501277120460624	0.20833333333333331
test.A.mA2()	0.2886751345948129	0.31896098681674395	Inf	Inf	0.0	0.2916666666666667	0.4246263530240907	Inf
test.A	0.041666666666666664	0.2886751345948129	Inf	Inf	0.2916666666666667	0.0	0.39965262694272663	Inf
test.B.mB1()	0.39747466725706065	0.42666573660612916	0.46955948623269356	0.43501277120460624	0.4246263530240907	0.39965262694272663	0.0	0.44095855184409843
test.B	Inf	Inf	0.2916666666666667	0.20833333333333331	Inf	Inf	0.44095855184409843	0.0

Результаты запуска

- Реальный пример: код FoodManager’a из прошлого семестра
- Рекомендуется перенести 39 методов (~16%)
- Создать 10 новых классов
- Часть из методов, которые предлагается передвинуть - методы из классов Activity



Результаты запуска

- Реальный пример: сетевая игра “Крестики-Нолики”
- Рекомендуется перенести 3 метода
- Создание и удаление классов не требуется

```
Move Method count: 3
Create Class count: 0
Remove Class count: 0
Method Count: 25
ClassCount: 6
```

```
Move method ru.spbau.svidchenko.cw03.net.Network.getClick() to class ru.spbau.svidchenko.cw03.net.Server
Move method ru.spbau.svidchenko.cw03.gui.GameGUI.receiveClick(int,int) to class ru.spbau.svidchenko.cw03.logic.GameLogic
Move method ru.spbau.svidchenko.cw03.gui.GameGUI.createGUI() to class ru.spbau.svidchenko.cw03.logic.GameLogic
```

Итоги

- Научился писать плагины для IDEA
- Разобрался в большом проекте
- Изучил статьи
- Реализовал метод поиска оптимального разбиения на классы
- Также реализовал 16 различных метрик
- https://github.com/ml-in-programming/MetricsReloaded