Бережных А.В.

Научный руководитель: к. т. н. Брыксин Т. А.

Эффективный поиск дубликатов в объемной кодовой базе в Intellij IDEA

Актуальность

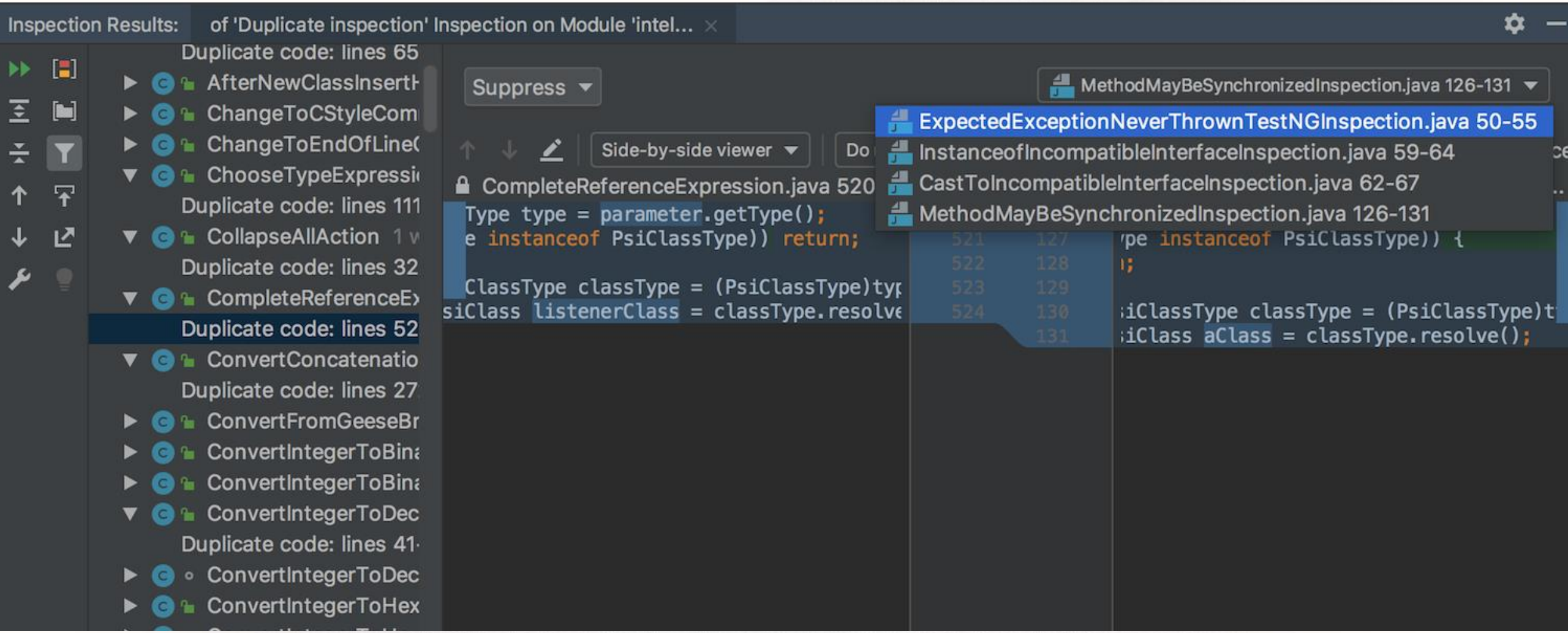
- Дублирование кода – одна из наиболее часто встречающихся проблем в разработке программного обеспечения
  - Тяжелее уловить разницу в повторяющихся участках кода и понять назначение того или иного его фрагмента
  - Любое изменение во фрагменте кода необходимо применить ко всем его дубликатам.
  - Время, потраченное на внесение изменений и последующее тестирование кода увеличивается пропорционально количеству дубликатов
- Редуцирование количества дубликатов повышает общее качество кода и упрощает его разработку и сопровождение

Актуальность

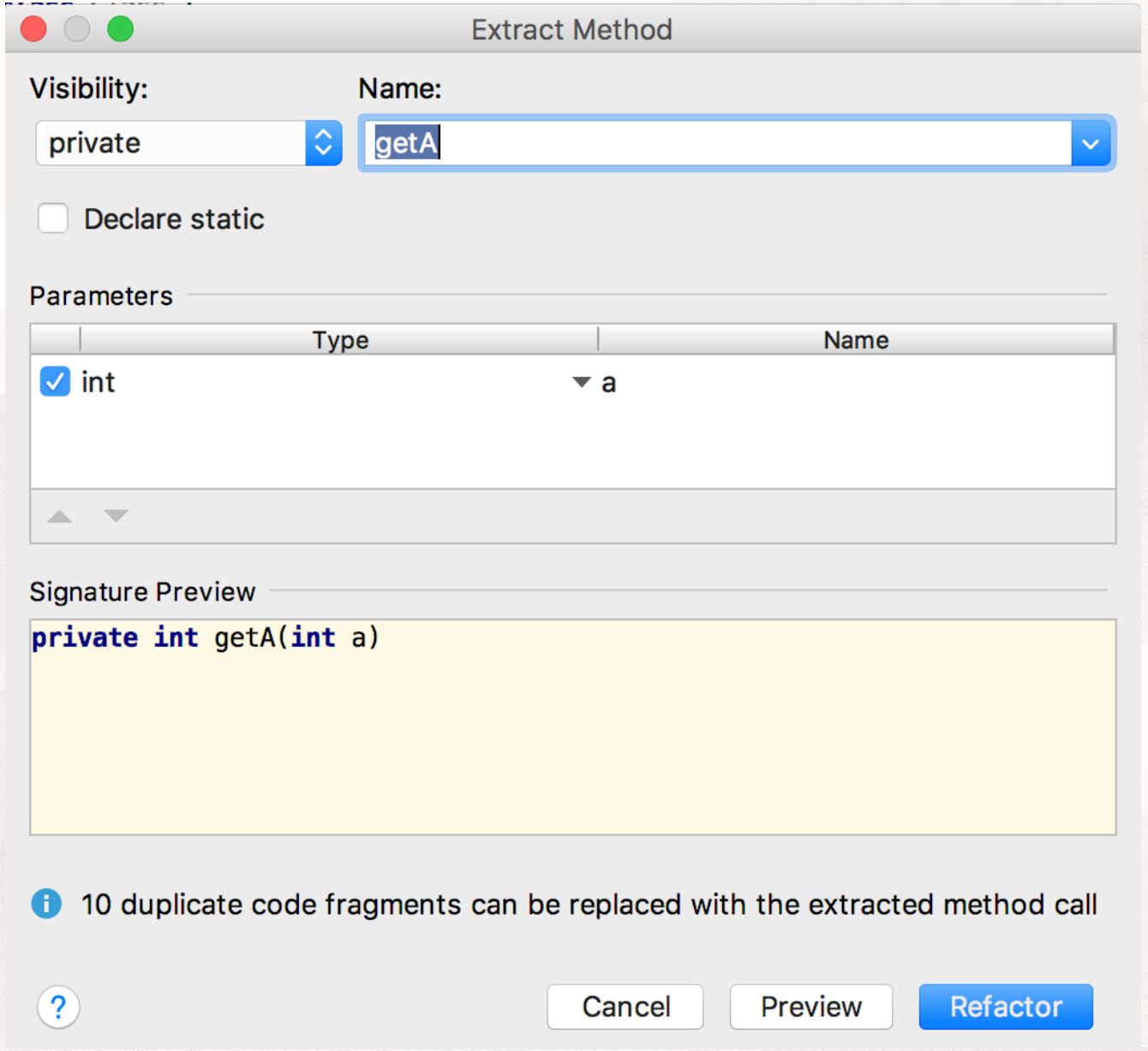
- В IntelliJ IDEA Ultimate встроены инструменты, позволяющие обнаруживать и устранять дублирующийся код
  - Инструмент поиска дубликатов на больших объёмах кода
  - Инструмент выделения кода дубликатов в метод

- Пользователю отображается полный список найденных групп дубликатов
  - Пользователю необходимо вручную для каждой группы проверять возможность выделения общего кода
  - К большей части групп невозможно применить инструмент выделения
  - Применение тривиального метода фильтрации на основе встроенного инструмента выделения неприемлемо по времени работы

Пример поиска дубликатов



Пример выделения общего кода



Актуальность

- Там, где медленно работают классические алгоритмы, показывают своё успешное применение методы машинного обучения
- Предлагается рассмотреть данную задачу с точки зрения машинного обучения как задачу бинарной классификации

Постановка задачи

Цель

- Найти эффективный способ фильтровать список выдаваемых дубликатов на возможность применения к ним инструмента выделения метода

Задачи

- Собрать данные, необходимые для обучения
- Выбрать и реализовать метод векторизации кода дубликатов
- Сравнить эффективность выбранного метода векторизации с аналогами

Сбор датасета

Создан инструмент для сбора данных о дубликатах кода в IntelliJ IDEA Ultimate[1]

Принцип работы инструмента

- Принимает на вход список найденных дубликатов duplicates.xml
- Подключается к окну IDEA с открытым в нём списком дубликатов
- Эмулирует действия пользователя
  - Выбирает группы дубликатов
  - Ожидает от IDEA информацию о возможности применения рефакторинга к выбранной группе
  - Отмечает для группы в duplicates.xml полученную информацию (1 – можно применить, 0 – нет)

1. https://github.com/DedSec256/Medooza.NET

Сбор датасета

С помощью инструмента собраны данные:

Название проекта	Обьём (Строк кода)	Всего групп дубликатов	Нельзя выделить	Можно выделить
LWJGL3	388199	3467	2505	962
LWJGL	72201	851	732	119
Junit5	68686	979	861	118
Jenkins	157361	1877	1464	413
Spring	642204	8718	6507	2211
RxJava	278931	5077	3352	1725
		20969	**15421**	**5548**

Векторизация фрагментов кода

[1]Свёртка AST на основе Tree-LSTM и word2vec

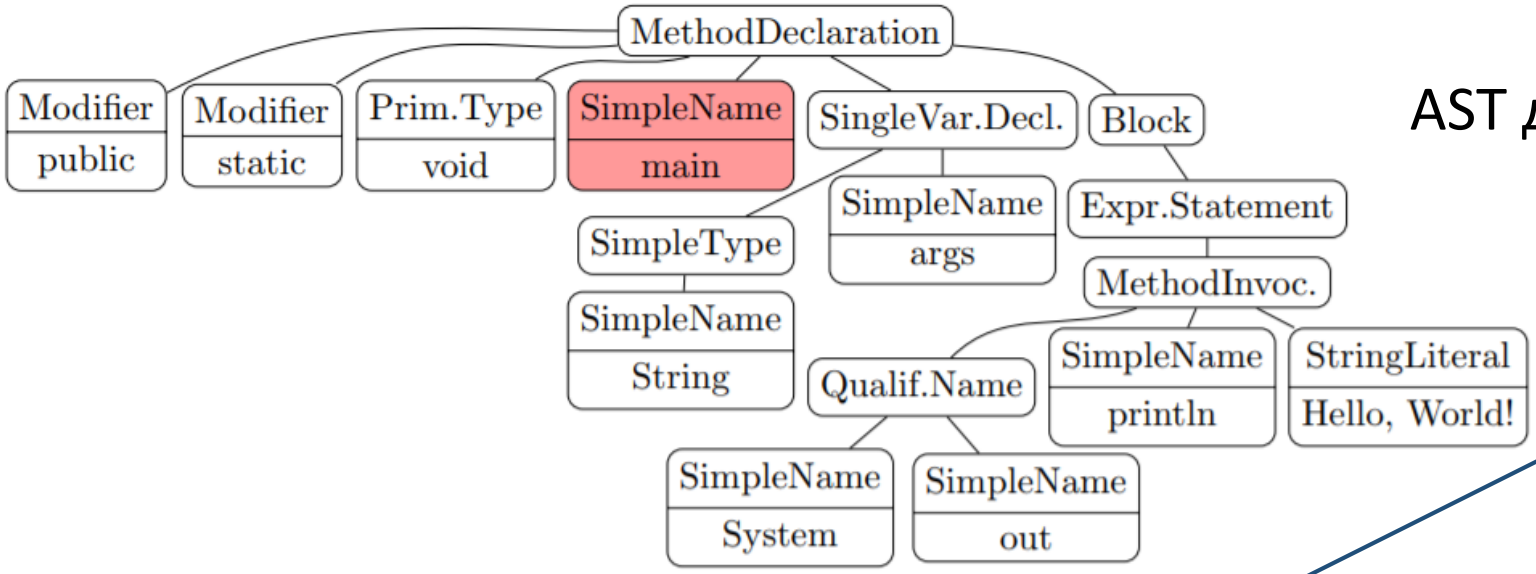
- Учитывает структуру кода и его внутренний контекст
- Демонстрирует хорошие результаты в задаче поиска дубликатов кода

1. L. Büch and A. Andrzejak. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 95–104, Feb 2019. doi: 10.1109/SANER.2019.8668039

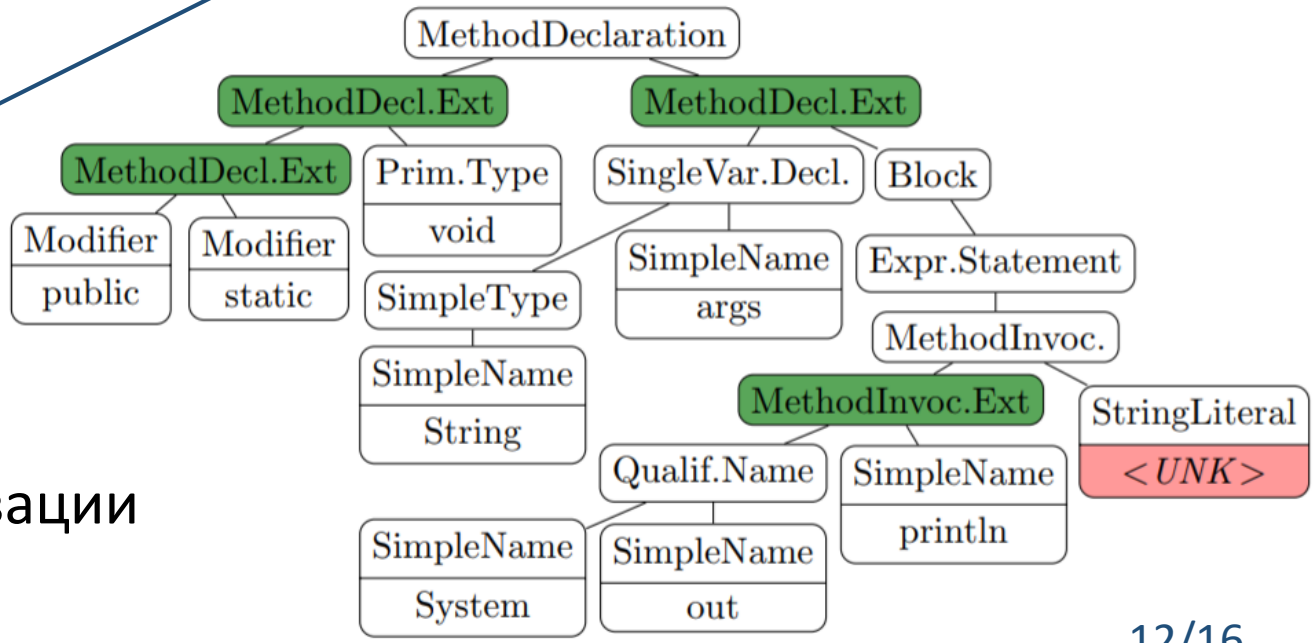
Векторизация фрагментов кода

- Получение AST из кода фрагмента с помощью Eclipse JDT
- Каждый тип и содержимое узла AST векторизуются с помощью word2vec
  - Из векторов составляется словарь
  - Размер векторного представления – 200
- Нормализация AST
  - Удаление идентификаторов методов и литералов
  - Бинаризация AST
  - Замена типа и содержимого каждого узла на его векторное представление из заранее подготовленного словаря
- Свёртка полученного дерева с помощью Tree-LSTM
  - Обучена авторами на датасете BigCloneBench

Пример нормализации AST



AST до нормализации



AST после нормализации

Процесс обучения

Фреймворк hyperopt-sklearn[1]

- Автоматический подбор гипер-параметров и выбор лучшего классификатора в процессе обучения
- Содержит 9 различных моделей классификации

Использован готовый инструментарий для обучения и оценки полученных результатов[2]

1. https://github.com/hyperopt/hyperopt-sklearn
2. https://github.com/MaxVortman/IDEA-code-clones

Используемые метрики

$$\text{FNR} = \frac{FN}{n}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{accuracy} = \frac{TP + TN}{n}$$

$$\text{FPR} = \frac{FP}{n}$$

$$\text{presicion} = \frac{TP}{TP + FP}$$

$$\text{ROC AUC} = \int_0^1 \gamma \, d\tau,$$

$$\gamma(\tau) = (\text{FPR}(\tau); \text{TPR}(\tau))$$

Где

- n – общее число групп дубликатов в датасете
- FN – группы, неверно помеченные моделью как не выделяемые
- FP – группы, неверно помеченные моделью как выделяемые
- TN – группы, верно помеченные моделью как не выделяемые
- TP – группы, верно помеченные моделью как выделяемые
- ROC AUC – площадь под кривой ошибок. Не зависит от выбранного порога

Оценка результатов

Для сравнения с результатами аналогов подобрано значение порога бинаризации так, чтобы значение FPR равнялось ~0.1

	threshold	FNR	FPR	accuracy	precision	recall	ROC AUC
tf-idf	0.5	0.19	0.1	0.71	0.76	0.62	0.8
bow	0.52	0.18	0.1	0.72	0.76	0.65	0.82
code2vec	0.51	0.17	0.1	0.72	0.76	0.66	0.83
ast-lstm	0.57	0.16	0.1	0.73	0.77	0.68	0.83
fastText	0.62	0.15	0.1	0.75	0.78	0.72	0.83

Результаты

- Реализован инструмент для сбора данных, необходимых для обучения
- Реализована техника векторизации кода дубликатов
- Посчитаны метрики и произведено сравнение с аналогами