

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК

Заведующий кафедрой, к.т.н., доцент


М. С. Воробьева

02.04. 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

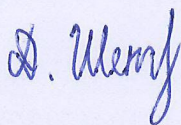
магистерская диссертация

РАЗРАБОТКА ВЕБ-СЕРВИСА ДЛЯ АНАЛИЗА ПРОГРАММНОГО КОДА
СТУДЕНТОВ ТЮМГУ

02.04.03 Математическое обеспечение и администрирование информационных
систем

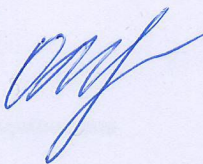
Магистерская программа «Разработка технологий Интернета вещей и больших
данных»

Выполнил работу
студент 2 курса
очной формы обучения



Шенгелия Давид Юзаевич

Научный руководитель
доцент кафедры
программного обеспечения



Иваненко Ольга Александровна

Рецензент
учебный мастер Института
математики и
компьютерных наук



Аврискин Михаил Владимирович

Тюмень
2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПО СТАТИЧЕСКОМУ АНАЛИЗУ КОДА	6
1.1. Количественные метрики	7
1.2. Метрики сложности потока управления программы	8
1.3. Индикатор качества ТЮВЕ.....	11
ГЛАВА 2. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ СТАТИЧЕСКОГО АНАЛИЗА КОДА.....	16
2.1. Cppcheck.....	16
2.2. PVS–Studio	16
2.3. Компилятор Roslyn (Roslyn Analyzers).....	17
2.4. Библиотека multimetric	18
ГЛАВА 3. АРХИТЕКТУРА СЕРВИСА ДЛЯ АНАЛИЗА ПРОГРАММНОГО КОДА СТУДЕНТОВ.....	22
3.1. Бизнес–процессы обработки исходных кодов студентов.....	22
3.1.1. Добавление кода студента в хранилище.....	22
3.1.2. Получение метрик кодов студентов.....	24
3.2. Модули анализа программного кода.....	25
3.3. Набор данных	25
3.4. Хранилище данных	27
ГЛАВА 4. ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ.....	33
ЗАКЛЮЧЕНИЕ	51
СПИСОК ЛИТЕРАТУРЫ.....	52

Приложение А. bash–скрипт для работы библиотеки multimetric	55
Приложение Б. API для работы с программным кодом.....	55
Приложение В. Реализация создания групп.....	60

ВВЕДЕНИЕ

Институт математики и компьютерных наук Тюменского государственного университета (ТюмГУ) состоит из множества направлений, связанных с программированием. В каждом из них неотъемлемой частью обучения является выполнение лабораторных и экзаменационных работ, сдача и защита курсовых работ, а также защита ВКР. Однако, несмотря на большое количество собранных данных по каждому студенту на период обучения, представленных в виде программного кода или содержащих его в выпускных работах, в ТюмГУ отсутствуют решения, позволяющие проанализировать каждую работу студента и получить на их основании информацию о том, по каким учебным модулям в различных дисциплинах студент показывает положительную динамику, а по каким ему необходимо развиваться, чтобы более детально разобрать материалы по дисциплине и сдать ее на «хорошо» или «отлично».

Исходя из вышеописанной проблемы была поставлена цель создания сервиса, позволяющего студенту на основании своих сданных работ в простом и понятном виде получить динамику собственной успеваемости, а также преподавателю определить эффективность работы групп и выявить показатели, по которым та или иная группа сильнее/слабее остальных. Для достижения этой цели поставлены следующие задачи:

- изучение существующих признаков для оценки программного кода;
- провести анализ имеющихся признаков для определения пороговых значений и их категоризации;
- создание веб-модуля для преподавателей с целью оценки программных кодов студентов по различным признакам.

Для подготовки и защиты выпускной квалификационной работы использовались поиск, анализ информации, системный подход для решения поставленных задач; приемы критического анализа проблемных ситуаций, а также средства и методы саморазвития и самореализации; методики межкультурного взаимодействия; умение расставлять приоритеты собственной

деятельности при работе в общем проекте в соответствии с командной стратегией для достижения поставленной цели.

Формулирование выводов по итогам проведенной работы осуществлялись с учетом применения современных коммуникативных технологий (в том числе на иностранном языке) для представления результатов на академических, профессиональных, экспертных ИТ-мероприятиях.

ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПО СТАТИЧЕСКОМУ АНАЛИЗУ КОДА

Для того, чтобы оценивать программный код по различным признакам, необходимо определить существующие на данный момент метрики [20] и выявить среди них ключевые, которые будут применены в дальнейшем.

Статический анализ кода представляет собой метод отладки компьютерной программы, который выполняется путем изучения кода без исполнения программы. При статическом анализе можно обнаружить много разнообразных дефектов и слабых мест исходного кода даже до того, как код будет готов для запуска. [10] Используя разные техники анализа, например, проверку абстрактного синтаксического дерева (AST) и анализ кодовых путей, инструменты статического анализа могут выявить скрытые уязвимости, логические ошибки, дефекты реализации и другие проблемы. Раннее обнаружение ошибок в коде особенно полезно для проектов больших встраиваемых систем, где разработчики не могут использовать средства динамического анализа до тех пор, пока программное обеспечение не будет завершено настолько, чтобы его было можно запустить на целевой системе. На этапе статического анализа обнаруживаются и описываются области исходного кода со слабыми местами, включая скрытые уязвимости, логические ошибки, дефекты реализации, некорректности при выполнении параллельных операций, редко возникающие граничные условия и многие другие проблемы. [18]

Инструменты статического анализа способны обнаруживать ошибки, которые пропускаются инструментами динамического анализа, потому что инструменты динамического анализа фиксируют ошибку лишь в случае, если во время тестирования ошибочный фрагмент кода исполняется.

В качестве первоначального исходного набора данных был использован архив с набором программных кодов студентов второго и третьего курсов направления МОиАИС из учебной платформы Moodle, написанных на языках программирования C# и C++.

1.1. Количественные метрики

Базовыми характеристиками исходного кода являются метрики, связанные с количеством рассматриваемых элементов (т.е. количественные характеристики). Данный класс метрик является простым для рассмотрения и при этом достаточным для общей оценки кода. Среди них можно выявить следующие метрики:

- **количество строк кода;**
- **доля строк с комментариями** — отношение количества строк, содержащих комментарии к общему числу строк (в процентах);
- **доля пустых строк** — отношение количества строк без символов к общему числу строк (в процентах);
- **средняя длина строки** — усредненное значение от количества символов в строке;
- **средняя длина идентификатора** — усредненное количество символов для описания переменных;
- **средняя длина блоков кода с учетом вложенности (*AvgNestedBlockLen* в формуле)** — отношение произведения максимальной вложенности блоков во всех методах (*maxNesting*) и количества символов в блоках всех методов (*symBlocksCnt*) к количеству методов (*N*):

$$AvgNestedBlockLen = \frac{\sum_{i=1}^n maxNesting(i) * \sum_{i=1}^n symBlocksCnt(i)}{N} \quad (1);$$

- **метрики Холстеда [14]**

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- n_1 — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов)
- n_2 — число уникальных операндов программы (словарь операндов)
- N_1 — общее число операторов в программе
- N_2 — общее число операндов в программе

В таблице 1 перечислены метрики, основанные на вышеописанных характеристиках.

Таблица 1

Метрики Холстеда

Параметр	Символ	Формула
длина программы	N	$N_1 + N_2$
словарь программы	n	$n_1 + n_2$
объем программы	V	$N * \log_2 n$
сложность управления программой	D	$\left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right)$
усилия для реализации алгоритма в программе	E	$D * V$
число Страуда	S	$5 \leq S \leq 20$ (для метрики Холстеда $S=18$)
время для реализации алгоритма	T	$\frac{E}{S}$

1.2. Метрики сложности потока управления программы

Данный класс метрик основан на анализе управляющего графа программы, который является ориентированным графом с одним входом и одним выходом. Вершинами графа являются участки программного кода с последовательными вычислениями и без операторов цикла и ветвления, дугами графа являются переходы от одного блока программы к другому, а также ветви выполнения программы.

Основной используемой метрикой такого класса является **цикломатическая сложность программы** (или цикломатическое число Мак–Кейба) [12], вычисляемая по формуле $V(G) = e - n + 2 * p$ (2), где e – число дуг, n – число вершин, p – количество компонентов связности.

На ее основе создано множество модификаций [20]:

- метод Майерса (использование интервала $[V(G), V(G) + h]$, где $h = 0$ для простых предикатов, а для n -уровневых $h = n - 1$);
- метод Хансена (использование пары мер сложности – цикломатическая сложность + число операторов);
- топологическая мера Чена (сложность программы, выраженная через количество пересечений границ между областями в графе программы).

На рисунках 1 и 2 показан код программы (пунктирной линией обозначены циклы и ветвления, сплошной линией – операторы присваивания) и соответствующий ему управляющий граф на примере проверки чисел в массиве на четность. В данном примере в графе 11 узлов, 14 ребер и 1 компонента связности. Согласно формуле 2, цикломатическая сложность в данном примере равна 5.

```

using System;

public class Program
{
    public static void Main()
    {
        int x = 0;
        int parity = 0;
        int[] a = new int[] {0,1,2,3,4,5,7,8,9,10};
        bool p = false;

        //(1)
        while (x<10) {
            //(2)
            if (a[x] % 2 == 0) {
                //(3)
                parity = 0;
            }
            else {
                //(4)
                parity = 1;
            }
            //(5)
        }
        //(6)
        switch(parity){
            case 0:
                //(7)
                Console.WriteLine(string.Format("a[{0}] - четное число", x));
                break;
            case 1:
                //(8)
                Console.WriteLine(string.Format("a[{0}] - нечетное число", x));
                break;
            default:
                //(9)
                Console.WriteLine("Ошибка");
                break;
        }
        //(10)
        x++;
    }
    //(11)
    p = true;
}
}

```

Рисунок 1. Пример кода программы с обозначением операторов

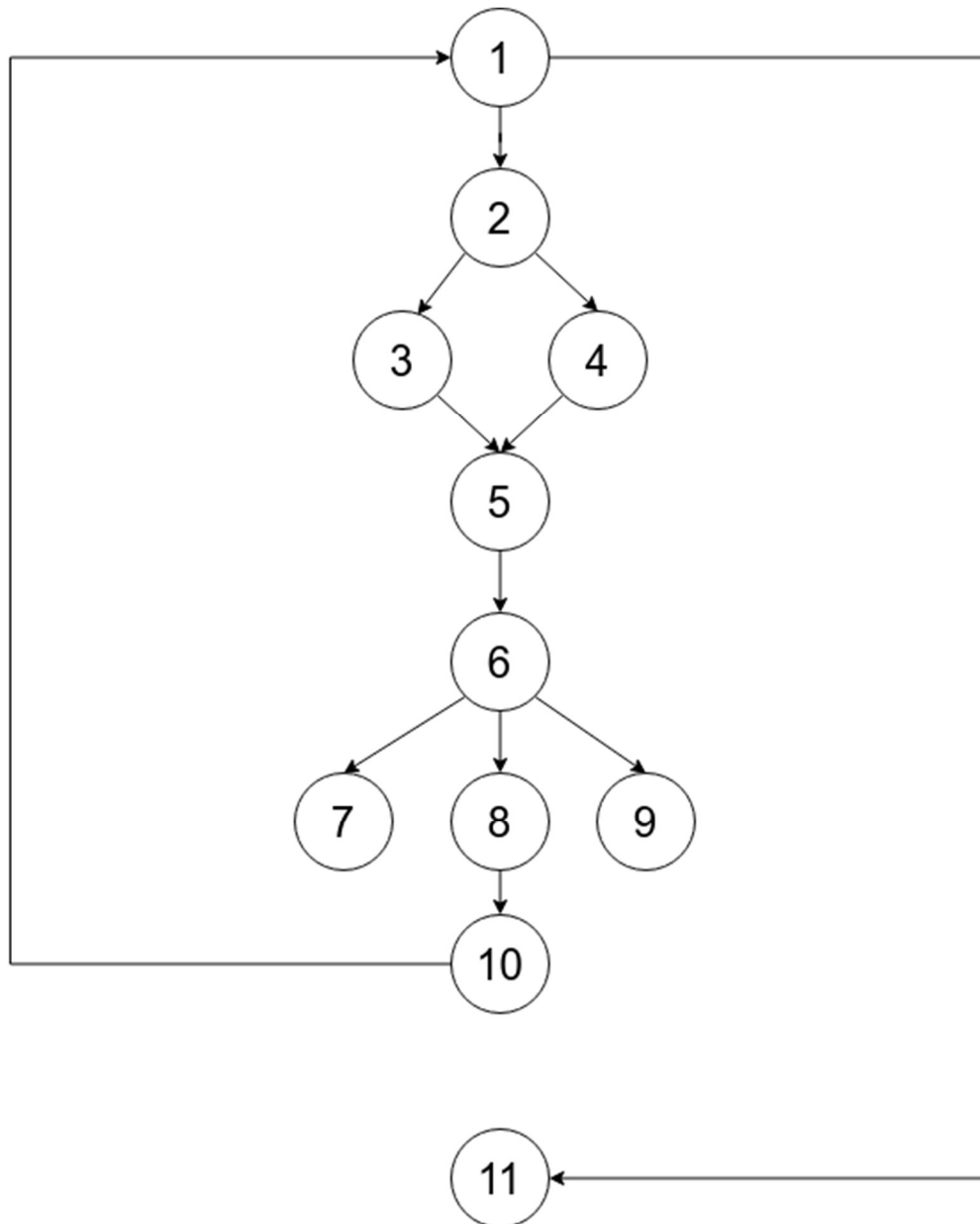


Рисунок 2. Управляющий граф программы

Несмотря на наличие возможных модификаций классического метода цикломатической сложности программы, в большинстве существующих инструментов используется именно цикломатическое число Мак–Кейба ввиду простоты и универсальности данной метрики.

1.3. Индикатор качества ТЮВЕ

Отдельным пунктом стоит отметить набор метрик, используемых компанией ТЮВЕ для оценки качества кода [8]. Они базируются на стандарте ISO 25010 [5], который определяет модель качества продукта, состоящую из 8 характеристик:

- **функциональная пригодность** (степень, в которой продукт обеспечивает функции, соответствующие заявленным и подразумеваемые потребности, когда продукт используется в определенных условиях);
- **уровень производительности** (производительность относительно количества ресурсов, используемых в указанных условиях);
- **совместимость** (степень, в которой две или более системы могут обмениваться информацией и/или выполнять свои требуемые функции, используя одну и ту же аппаратную или программную среду);
- **удобство использования** (юзабилити);
- **надежность** (степень, в которой система или компонент выполняет определенные функции в определенных условиях в течение определенного периода времени);
- **защищенность** (степень защиты информации и данных, чтобы неавторизованные лица или системы не могли их прочитать или изменить, а уполномоченным лицам или системам не было отказано в доступе к ним);
- **сопровождаемость** (степень эффективности модифицируемости продукта);
- **переносимость** (степень, в которой система может быть эффективно и действенно перенесена с одного оборудования, программного обеспечения или другой операционной среды или среды использования в другую).

Рассмотрим те метрики, которые не были указаны в предыдущих пунктах, и которые можно реализовать в рамках статического анализа кода:

Покрытие кода

Измеряется как среднее из доступных значений покрытия решений, ветвей и операторов. Должен быть доступен хотя бы один из этих трех типов покрытия. Данная метрика основана на исследовании Стива Корнетта [1]. Покрытие функций не принимается во внимание, поскольку для этого типа покрытия слишком легко достичь высоких показателей покрытия кода.

Формула для расчета оценки покрытия кода:

$$score = \min(0.75 * testcoverage + 32.5, 100) \quad (3)$$

На рисунке 3 показана градация по категориям в зависимости от качества покрытия программного кода.

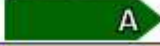





Code Coverage	TQI Score	Category
>= 76.7%	>= 90%	 A
>= 63.3%	>= 80%	 B
>= 50%	>= 70%	 C
>= 23.3%	>= 50%	 D
>= 10%	>= 40%	 E
< 10%	< 40%	 F

Рисунок 3. Категории покрытия программного кода

Дублирование программного кода

Данная метрика рассчитывается путем подсчета количества семантически эквивалентных цепочек из 100 токенов. Токен – это атомарный строительный блок языка программирования. Примеры токенов: идентификаторы (например, «id»), ключевые слова (например, «return»), операторы (например, «&&») и разделители (например, «{» или «;»). Общее количество токенов, содержащих дублированную цепочку, берется и выражается в процентах от общего количества токенов в системе. При расчёте не учитываются такие цепочки токенов, как: комментарии, отступы, заголовочные файлы C/C++ и директивы using в C#.

Формула для расчета оценки дублирования программного кода:

$$score = \min(-40 * \lg(codeduplication) + 80, 100) \quad (4)$$

На рисунке 4 показана градация по категориям в зависимости от степени дублирования программного кода.







Code Duplication	TQI Score	Category
<= 0.56%	>= 90%	 A
<= 1.00%	>= 80%	 B
<= 1.78%	>= 70%	 C
<= 5.62%	>= 50%	 D
<= 10.00%	>= 40%	 E
> 10.00%	< 40%	 F

Рисунок 4. Категория дублирования программного кода

Разветвление в проекте (Fan Out)

Метрика Fan Out [4] измеряется путем подсчета среднего количества операций импорта на модуль. Данная метрика зависит от языка программирования. Для С и С++ используется количество операторов include, для Java – количество операторов import. Для С# метрика является более комплексной, т.к. в С# используется другой механизм импорта. Оператор using в С# импортирует полное пространство имен, которое может состоять из сотни классов, тогда как на самом деле используются лишь некоторые из них.

Также существует разница между внешним Fan Out и внутренним. Внешний Fan Out касается импорта извне программной системы, тогда как внутренний касается ссылок внутри самого проекта. Внешний импорт в основном применяется для повторного использования существующего программного обеспечения и поэтому намного лучше, чем внутренний импорт. Следовательно, внутренний импорт оказывает в 4 раза большее негативное влияние на TQI для Fan Out, чем внешний импорт. Среднее значение Fan Out в программном коде отображается в нормативной шкале с помощью следующей формулы:

$$score = \frac{100}{\frac{8 * internalfanout}{2} + \frac{*externalfanout}{100}} \quad (5)$$

На рисунке 5 показана градация по категориям в зависимости от показателя Fan Out.

Fan Out	TQI Score	Category
<= 3.04	>= 90%	A
<= 6.44	>= 80%	B
<= 10.29	>= 70%	C
<= 20.00	>= 50%	D
<= 26.43	>= 40%	E
> 26.43	< 40%	F

Рисунок 5. Категория метрики Fan Out.

Предупреждения компилятора

Данная метрика измеряется путем запуска используемого компилятора на максимально возможном уровне предупреждений. Если используется более одного компилятора (например, потому что код создается для нескольких платформ), предупреждения всех компиляторов объединяются. Если файл не работает для одного из компиляторов в случае использования нескольких, то штрафа к оценке не будет, при условии, что файл компилируется, по крайней мере, для одного из компиляторов.

Поскольку разные компиляторы проверяют наличие разных предупреждений компилятора, недостаточно использовать количество предупреждений компилятора в качестве входных данных для оценки. Следовательно, набор предупреждений компилятора должен быть нормализован в зависимости от количества различных проверок, выполняемых компилятором, и серьезности этих проверок. Для этого ТЮВЕ использует свой коэффициент соответствия, который представляет собой число от 0 (нет соответствия) до 100 (полное соответствие, то есть без предупреждений компилятора).

Как только коэффициент соответствия известен, применяется следующая формула для определения оценки предупреждений компилятора:

$$\text{score} = \max(100 - 50 * \lg(101 - \text{co_factor}(\text{compiler_warnings})), 0) \quad (6)$$

На рисунке 6 показана градация по категориям в зависимости от показателя метрики предупреждения компиляции.

Compliance Factor	TQI Score	Category
>= 99.42%	>= 90%	A
>= 98.49%	>= 80%	B
>= 97.02%	>= 70%	C
>= 91.00%	>= 50%	D
>= 85.15%	>= 40%	E
< 85.15%	< 40%	F

Рисунок 6. Категория метрики предупреждения компиляции.

ГЛАВА 2. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ СТАТИЧЕСКОГО АНАЛИЗА КОДА

2.1. Cppcheck

Cppcheck – это инструмент статического анализа кода, написанного на языках C/C++. Цель данного инструмента состоит в том, чтобы уменьшить количество ложных срабатываний. Он обеспечивает уникальный анализ кода для обнаружения ошибок и фокусируется на обнаружении неопределенного поведения (*undefined behaviour*) и потенциально опасных конструкций в участках кода [2]. Cppcheck предназначен для анализа кода C/C++, даже если он имеет нестандартный синтаксис (распространенный во встроенных проектах), для этого предназначена возможность указания ссылки на сторонние библиотеки, а также на собственные модули в проекте. Преимуществом данного инструмента является интеграция с множеством существующих средств разработки как в нативном виде (Buildbot, Code::Blocks, CodeDX и т.д.), так и в виде плагина (Visual Studio, QtCreator, Eclipse и т.д.). Недостатком является возможность анализа только двух языков программирования – C/C++, а также чрезмерная зависимость полученных метрик от указания ссылок на сторонние библиотеки, что негативно влияет на возможность автономного (без возможности получения ссылок на библиотеки, когда код дан исключительно в виде текстового файла) тестирования кода.

2.2. PVS-Studio

PVS-Studio – это инструмент для выявления ошибок и потенциальных уязвимостей в исходном коде программ, написанных на языках C, C++, C# и Java. Работает в 64-битных системах на Windows, Linux и macOS и может анализировать код, предназначенный для 32-битных, 64-битных и встраиваемых ARM платформ [16]. Данный инструмент обладает довольно большой базой ошибок, обнаруженных в open-source проектах [19], а также имеет множество технологий при анализе компилируемого кода, а именно:

- сопоставление с шаблоном (pattern-based analysis) на основе абстрактного синтаксического дерева для поиска мест в исходном коде, которые похожи на известные шаблоны кода с ошибкой;
- вывод типов (type inference) на основе семантической модели программы для получения полной информации о всех переменных и выражениях, встречающихся в коде;
- символьное выполнение (symbolic execution) для вычисления значений переменных, которые могут приводить к ошибкам, проведение проверки диапазонов (range checking) значений;
- анализ потока данных (data-flow analysis) для вычисления ограничений, накладываемых на значения переменных при обработке различных конструкций языка. Например, какие значения может принимать переменная внутри блоков if/else;
- аннотирование методов (method annotations), предоставляющее больше информации об используемых методах, чем может быть получено путем анализа только их сигнатуры.

Основным недостатком данного инструмента является его проприетарность и платность (доступны только Team License и Enterprise License). Также недостатком является генерация ложноположительных сообщений при корректном коде (пример: ключевое слово auto может обозначать возвращаемый тип void, однако PVS-Studio выдаёт ошибку вида «V591 Non-void function should return a value», подразумевая, что auto возвращает исключительно непустые значения).

2.3. Компилятор Roslyn (Roslyn Analyzers)

Roslyn – это платформа с открытым исходным кодом, разработанная Microsoft и содержащая компиляторы и инструменты для синтаксического анализа и анализа кода, написанного на C# и Visual Basic. Roslyn используется в среде Microsoft Visual Studio. С помощью компилятора Roslyn (и анализатора Microsoft.CodeAnalysis.NetAnalyzers из Roslyn Analyzers в частности) [6]

реализованы различные нововведения, такие как анализ кода и его исправление на основании проведенного анализа. Используя инструменты анализа, предоставляемые Roslyn, можно выполнить полный синтаксический анализ кода, проанализировав все поддерживаемые языковые конструкции.

Синтаксическое дерево является базовым элементом для анализа кода, т.к. по нему происходит перемещение в ходе анализа. Дерево строится на основе кода, приведенного в файле, следовательно каждый файл имеет свое синтаксическое дерево, которое является неизменяемым. Для изменения кода требуется новое синтаксическое дерево, которое может быть построено на основе первоначального [15]. Существует 3 основных элемента синтаксического дерева:

- **Syntax nodes** – одни из основных элементов дерева, представляют собой синтаксические конструкции языка. К этой категории относятся определения, объявления, операторы, выражения и т.п.;
- **Syntax tokens** – терминалы грамматики языка (лексемы, конечные символы), представляющие наименьшие синтаксические фрагменты кода. Они никогда не являются родителями других узлов. К этой категории относятся ключевые слова, идентификаторы, специальные символы и т.п.;
- **Syntax trivia** – дополнительная синтаксическая информация, которые в значительной степени несущественны для нормального понимания кода. Данные узлы не входят в конечную компиляцию кода. К этой категории относятся пробелы, переводы строк, комментарии, директивы препроцессора и т.п.

2.4. Библиотека *multimetric*

Данная библиотека, написанная на языке Python, позволяет посчитать различные метрики программного кода для множества языков программирования (C/C++/C#/Java, Python, JS, и т.д.), а именно [7]:

- Процент комментариев в коде.

- Цикломатическая сложность по Мак–Кейбу.
- Метрики Холстеда (сложность, усилия, объем).
- Индекс пригодности кода к доработке.
- Метрика согласно pylint.
- Метрики согласно ПЛОВЕ.

В таблице 2 указано более детальное описание параметров возвращаемого результата вычисления в виде json–структуры.

Таблица 2

Структура возвращаемого результата

Наименование	Описание	Диапазон значений	Рекомендуемые значения
comment_ratio	Отношение количества комментариев к коду (в %)	0..100	> 30.0
cyclomatic_complexity	Цикломатическая сложность по МакКейбу	0..(inf)	< 10
fanout_external	Количество импортированных модулей вне исходного дерева (проекта)	0..(inf)	
fanout_internal	Количество импортированных модулей из одного исходного дерева (проекта)	0..(inf)	
halstead_bugprop	Количество ошибок по метрике Холстеда	0..(inf)	< 0.05
halstead_difficulty	Сложность по метрике Холстеда	0..(inf)	
halstead_effort	Усилия по метрике Холстеда	0..(inf)	
halstead_timerequired	Время, необходимое для программирования по метрике Холстеда	0..(inf)	
halstead_volume	Объем работы по метрике Холстеда	0..(inf)	

lang	Список определенных языков программирования из списка файлов	list	
loc	Кол-во строк кода	1..(inf)	
maintainability_index	Индекс поддерживаемости проекта	0..100	> 80.0
operands_sum	Количество используемых операндов	1..(inf)	
operands_uniq	Количество уникальных используемых операндов	1..(inf)	
operators_sum	Количество используемых операторов	1..(inf)	
operators_uniq	Количество уникальных используемых операторов	1..(inf)	
pylint	Общая оценка качества согласно pylint	0..100	> 80.0
tiobe_compiler	Оценка предупреждений компилятора согласно метрике TIOBE	0..100	> 90.0
tiobe_complexity	Сложность согласно метрике TIOBE	0..100	> 80.0
tiobe_coverage	Покрытие кода согласно метрике TIOBE	0..100	> 80.0
tiobe_duplication	Оценка дублирования кода согласно метрике TIOBE	0..100	> 80.0
tiobe_fanout	Оценка Fan-Out (см. fanout_external и fanout_internal) по версии TIOBE	0..100	> 80.0
tiobe_functional	Оценка функциональных дефектов по метрике TIOBE	0..100	> 90.0
tiobe_security	Оценка безопасности согласно метрике TIOBE	0..100	> 90.0

Продолжение таблицы 2

tiobe_standard	Оценка стандартизации кода по языку согласно TIOBE	0..100	> 80.0
tiobe	Общая оценка качества согласно TIOBE	0..100	> 80.0

Преимуществами данного инструмента является возможность автономной проверки качества кода, большой список анализируемых языков программирования, а также большой набор метрик и использование алгоритмов, доступных в свободном использовании.

ГЛАВА 3. АРХИТЕКТУРА СЕРВИСА ДЛЯ АНАЛИЗА ПРОГРАММНОГО КОДА СТУДЕНТОВ

3.1. Бизнес–процессы обработки исходных кодов студентов

3.1.1. Добавление кода студента в хранилище

Со стороны студента или преподавателя через веб–интерфейс вносится выполненное задание, после чего API передается POST–запрос на добавление текстового файла с выполненной задачей. Со стороны API происходит генерация уникального идентификатора в guid–формате с последующим добавлением записи в PostgreSQL с указанием идентификатора, датой работы и преподавателем. Также, API предоставляет структурированные данные для их последующей обработки в модуле статического анализа кода, где происходит токенизация текстовых данных. Полученный список токенов и соответствующих им данных используется в том же модуле для расчета метрик программного кода. После этого API получает результат расчета в json–формате и добавляет его в HBase вместе с исходным текстом. После этого через API в веб–интерфейс отображается уведомление об успешной обработке записи. Визуализация данного процесса показана на рисунке 7.

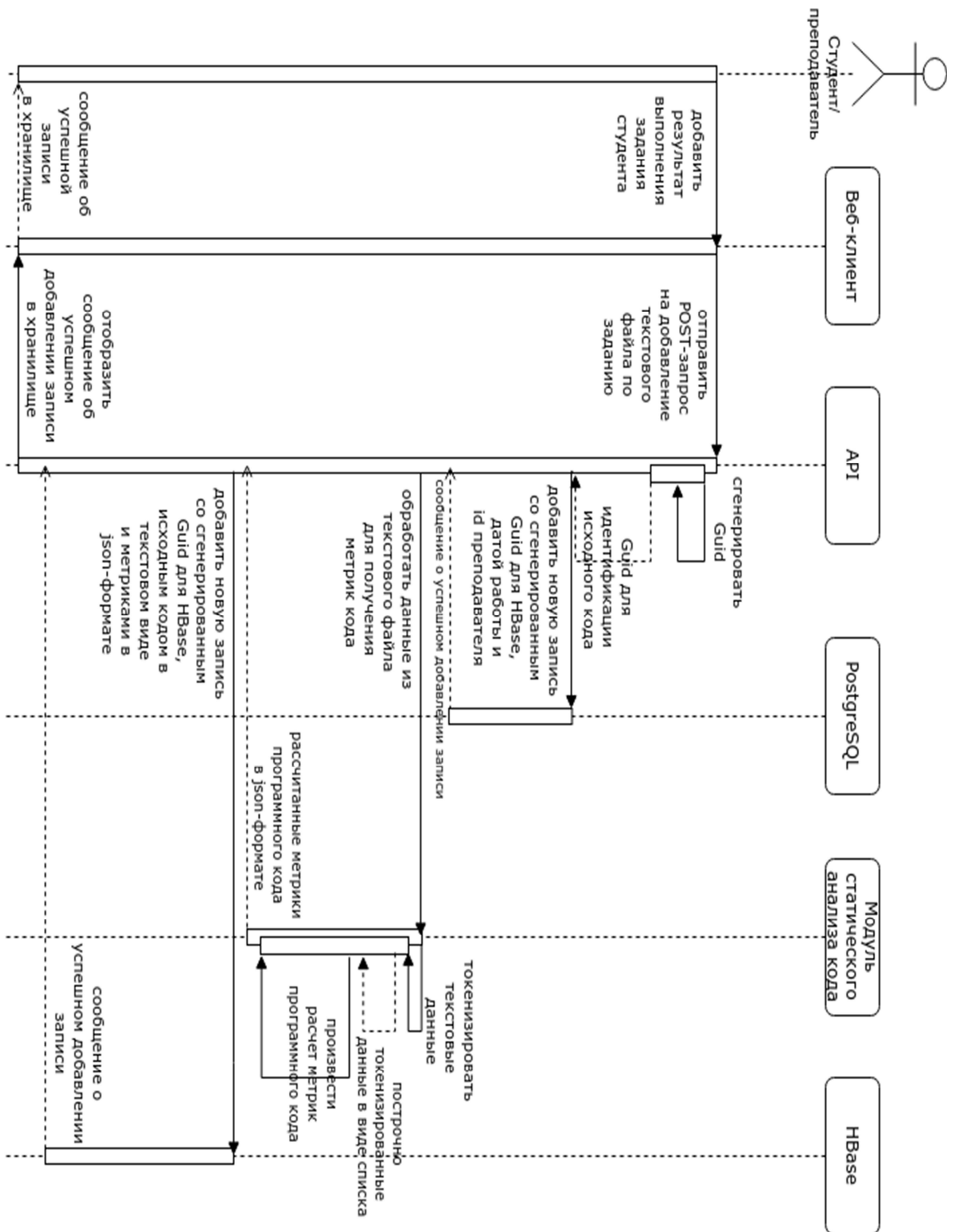


Рисунок 7. Бизнес–процесс добавления программного кода студента в хранилище в виде диаграммы последовательностей UML

3.1.2. Получение метрик кодов студентов

Со стороны преподавателя через веб-интерфейс к API передается GET-запрос на получение метрик кодов студентов по выбранному заданию, затем API обрабатывает получение метаданных кодов студента из таблицы `code_metas`, а именно идентификатор записи, дата сдачи и другие параметры. После этого на уровне API происходит обращение к хранилищу HBase для получения рассчитанных метрик кодов в json-формате. При получении данных из HBase происходит декодирование результата из массива байт в строковое значение. Визуализация данного процесса показана на рисунке 8.

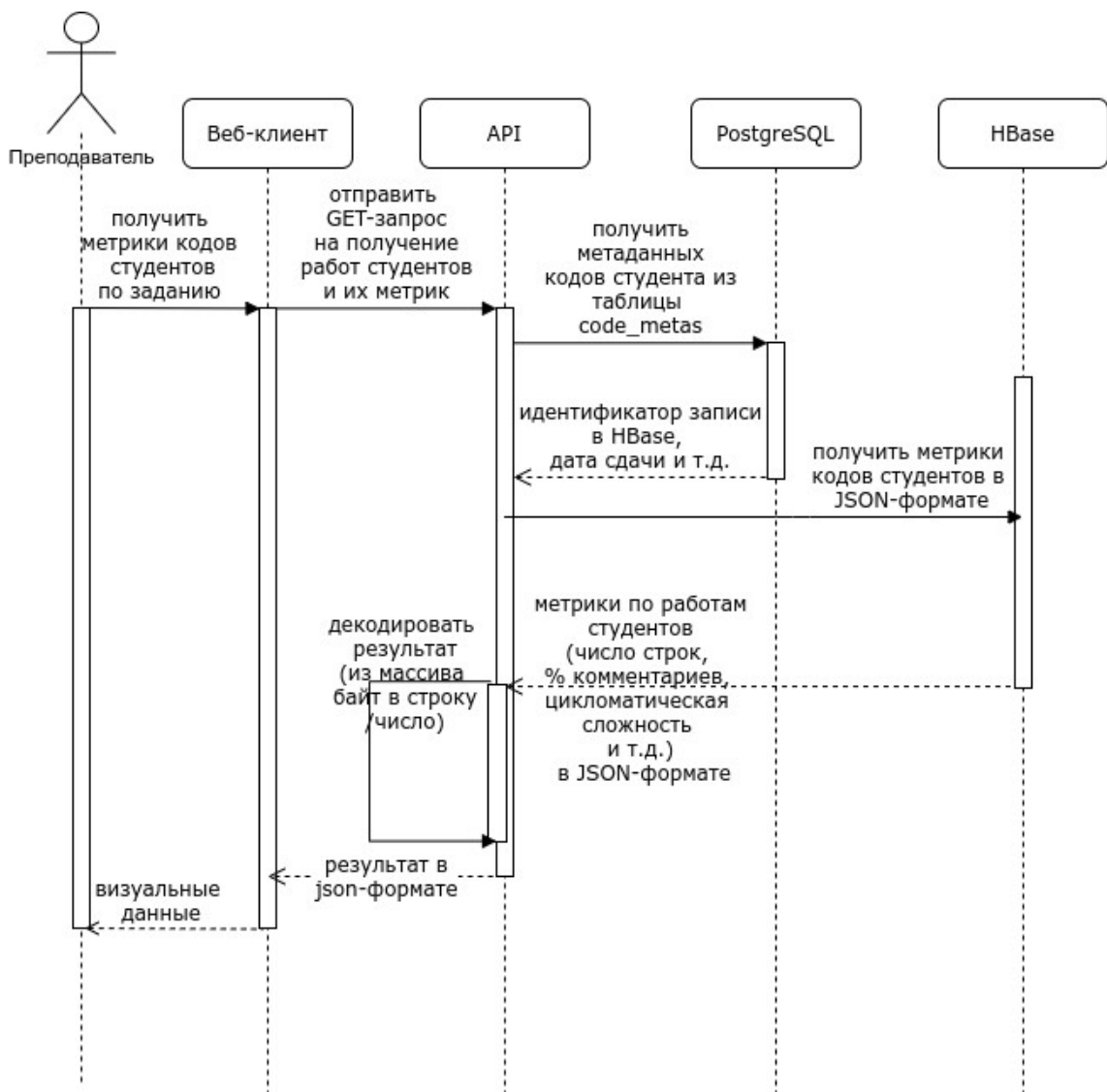


Рисунок 8. Бизнес-процесс получения метрик кодов студентов в хранилище в виде диаграммы последовательностей UML

3.2. Модули анализа программного кода

Ключевым этапом для статического анализа программного кода является токенизация исходного текста (определение ключевых слов для каждого из типов анализа программного кода) [3] для дальнейшего расчета метрик на основании формул, связанных с количеством ключевых слов и их типов/видов в зависимости от типа текстового токена. Например, для оценки цикломатической сложности программного кода в качестве токенов используются слова, описывающие условные операторы («if», «else», «case», «&&» и т.д.) и операторы выхода из цикла/функции («return», «exit») или продолжения цикла («continue», «yield»), в то время как для оценки расчетных метрик (например, метрики Холстеда) используются токены операндов (переменных, классов, атрибутов, методов и т.д.) и токены операторов (разделителей, комментариев и т.д.). В качестве решения для реализации токенизации текста была использована библиотека `pygments` [17], отвечающая за подсветку синтаксиса в тексте на множестве различных языков программирования, в частности, был использован модуль `lexers`, предназначенный для разбиения входной последовательности символов на лексемы и последующей их классификации и унификации.

3.3. Набор данных

Для того, чтобы определить пороговые значения для каждой из метрик и категоризировать их, необходимо использовать большой набор текстовых данных с исходными кодами программ. В качестве такого набора был выбран датасет, сформированный сайтом `HackerRank`, который посвящен практическому программированию с исходными кодами на языке `C#` по 22 заданиям (сортировка вставками, быстрая сортировка, алгоритм составления большой суммы чисел, упрощенная версия задачи по игре в шахматы и т.д.) в количестве 329927 экземпляров [9].

Первоначально были взяты решения, прошедшие юнит-тестирование в рамках поставленной для них задачи, после чего была произведена их предобработка. Первый этап предобработки – обезличивание текстовых данных

(был сформирован серийный номер вместо имени пользователя, решившего задачу), далее – перенос и сохранение этих решений в папку `origin` (формат: `task_name/origin`). Вторым этапом является очистка сохраненных ранее данных, а именно: исключение упоминаний использования используемых сторонних библиотек путем сопоставления с частотным словарем ключевых слов, операторов, операндов и прочих символов, а также очистка от символов, полученных в результате возможных ошибок обработки и сохранения данных, таких как: маркеры последовательности байт (BOM) [13], лишние пробелы/разделители и символы из кодировок, отличающихся от UTF-8. После этого данные были перенесены и сохранены в папку `cleaned` (формат: `task_name/cleaned`). Третьим и последним этапом предобработки стало приведение данных, полученных на предыдущем этапе, к единому формату путем разделения текста на элементарные выражения (символы, ключевые слова, пространства имен и т.д.) при помощи сопоставления с ранее описанным частотным словарем и сохранение в папку `reduced` (формат: `task_name/reduced`). На рисунке 9 показана упрощенная схема обработки данных.

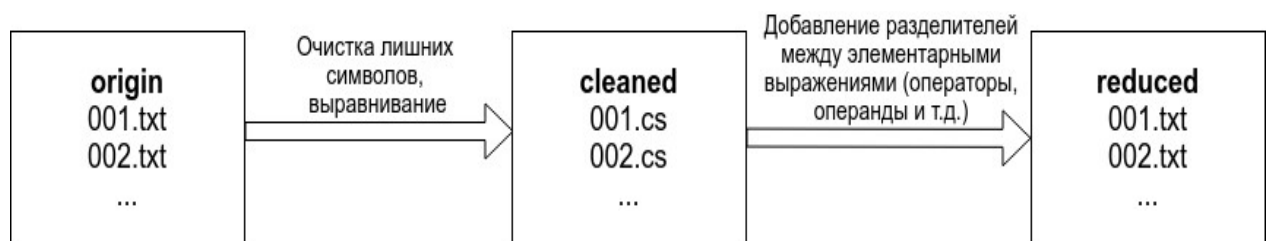


Рисунок 9. Последовательность действий обработки исходных кодов из ресурса HackerRank.

Данный датасет первоначально был предназначен для поиска дублирующих решений от разных пользователей (в нем также находится рассчитанное векторное пространство по всем решениям, сформированное из данных, полученных на третьем этапе, а также в каждой из задач расположены 3 csv-файла с 2 столбцами: первый столбец – имя файла, второй столбец – набор имен файлов с идентичной структурой), однако для задачи анализа программного кода были использованы данные из второго этапа, т.к. они являются наиболее релевантными набору данных студентов ТюмГУ.

3.4. Хранилище данных

В качестве хранилища для работы с программными кодами студентов различных групп используется набор Docker–контейнеров, в котором расположено несколько приложений, а именно:

- реляционная СУБД PostgreSQL для хранения основной информации о студентах и метаданных (схема представлена на рисунке 10);
- СУБД класса NoSQL Apache HBase для хранения текстовых данных больших размеров (УМК, курсовые/дипломные работы и т.д.).

Из общей схемы хранилища для анализа программного кода используются таблицы `students_info` для получения информации по студентам, `groups` для определения группы и направления, в которой обучается студент, `code metas` для получения метаданных о работах студентов по задачам, `tasks` для получения информации о лабораторных работах, которые назначают преподаватели, `teachers` и `teachers_groups` для получения ФИО и кафедры преподавателя и `subjs` для получения информации о дисциплинах (см. таблицы 3–9). Из Apache HBase используются таблицы `codes` для получения исходного кода и `numchars_code` для получения метрик по исходному коду (см. таблицы 10 и 11).

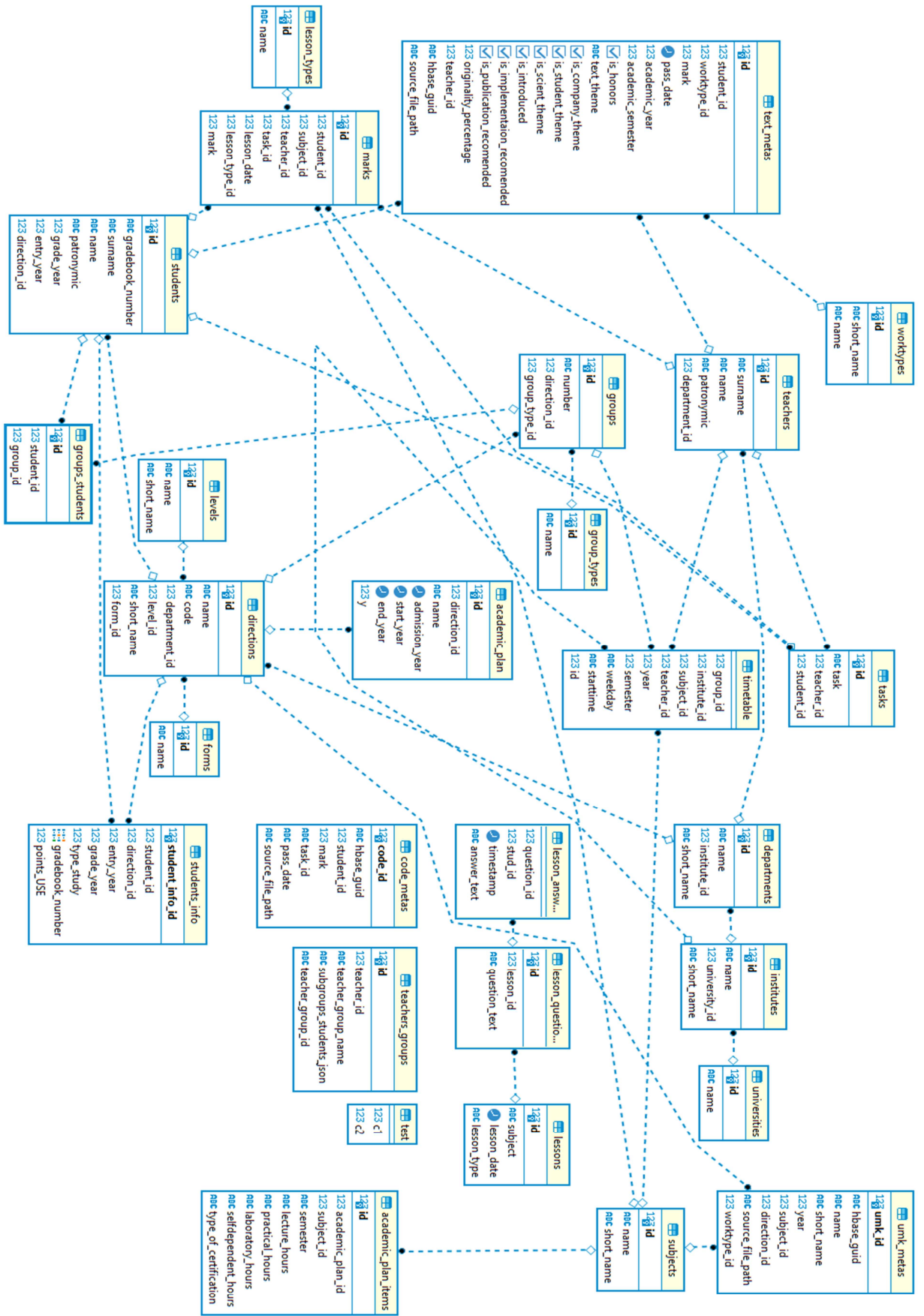


Рисунок 10. Реляционная часть хранилища

Описание таблицы students_info (Студенты)

№ п/п	Поля	Тип данных	Описание
1	student_info_id	bigint	id записи о студенте
2	student_id	bigint	id студента
3	direction_id	bigint	id направления
4	entry_year	varchar(255)	Год поступления
5	grade_year	varchar(255)	Год выпуска
6	type_study	bigint	тип обучения
7	gradebook_number	int	Номер зачетной книжки студента
8	email	int	адрес эл почты студента
9	points_USE	int	баллы за ЕГЭ

Описание таблицы groups (Группы)

№ п/п	Поля	Тип данных	Описание
1	id	bigint	id группы
2	number	varchar(255)	Номер группы
3	direction_id	bigint	id направления
4	group_type_id	bigint	id типа группы
5	start_date	timestamp	дата начала обучения в команде
6	end_date	timestamp	дата окончания обучения в команде

Описание таблицы code_metras (Метаданные кода программы)

№ п/п	Поля	Тип данных	Описание
1	id	bigint	id кода
2	hbase_guid	bigint	GUID текста
3	student_id	varchar(255)	id студента
4	mark	bigint	Оценка/баллы
5	task_id	bigint	id задания
6	pass_date	timestamp	Дата сдачи работы
7	source_file_path	varchar(255)	Путь к исходному файлу
8	task_number	timestamp	Номер задания

Описание таблицы tasks (Задания по предметам)

№ п/п	Поля	Тип данных	Описание
1	task_id	bigint	Id задания
2	task_text	varchar(255)	Текст задания
3	teacher_id	bigint	Преподаватель
4	subj_id	bigint	Дисциплина

Описание таблицы teachers (Преподаватели)

№ п/п	Поля	Тип данных	Описание
1	teacher_id	bigint	Id преподавателя
2	surname	varchar(255)	Фамилия
3	name	varchar(255)	Имя
4	patronymic	varchar(255)	Отчество
5	dep_id	bigint	id кафедры

Описание таблицы teachers_groups (Группировки преподавателей)

№ п/п	Поля	Тип данных	Описание
1	id	bigint	id группировки
2	teacher_id	varchar(255)	Id преподавателя
3	teacher_group_name	varchar(255)	наименование группы
4	subgroups_students_json	varchar(255)	Набор подгрупп
5	teacher_group_id	bigint	

Описание таблицы subjects (Дисциплины)

№ п/п	Поля	Тип данных	Описание
1	subj_id	bigint	id уровня
2	subj_name	varchar(255)	Наименование дисциплины
3	short_name	varchar(255)	Краткое наименование

Таблица 10

Описание таблицы codes (Исходные коды программ)

№ п/п	Поля	Тип данных	Описание
1	code	text (bytearray)	Исходный текст кода

Таблица 11

Описание таблицы numchars_code (Числовые характеристики программного кода)

№ п/п	Поля	Тип данных	Описание
1	guid	guid	Идентификатор исходного кода
2	metrics_json	text (bytearray)	Метрики программного кода в json-формате

ГЛАВА 4. ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ

Для реализации веб-модуля анализа программного кода было использовано несколько инструментов, а именно:

- компилятор Roslyn [11] из платформы .NET Core 3.1 SDK для анализа программного кода, написанного на языке C# (и опционально C++ без файлов заголовков);
- библиотека multimetric на языке Python для анализа программного кода на остальных языках программирования (C++, JavaScript, Python и т.д.).

Для проверки программного кода был использован набор данных, описанный в одноименном пункте из главы «Архитектура». Для каждого вида задания из выбранного набора данных было проведено измерение метрик программного кода (основные – цикломатическая сложность, объем и сложность по Холстеду и индекс поддерживаемости) и на основании этих данных были рассчитаны 2 вида статистических показателей: получение минимума, максимума и медианного значения метрик (расчет на примере задачи «Игра в шахматы» показан в таблице 12), а также расчет частотных показателей на основании разделения выборки по значениям метрик с шагом дискретизации, равном разности между максимальным и минимальным значением, деленной на 10 и округленной в большую сторону (пример расчета сложности по Холстеду и индекса поддерживаемости по аналогичной задаче показан в таблицах 13 и 14 соответственно и на рисунках 11 и 12 в виде графика распределения).

Таблица 12

Статистические характеристики метрик (минимум, максимум, медиана)

Метрика	min	max	median
cyclomatic_complexity	2	18	5
halstead_volume	54	1132	235.43
halstead_difficulty	0.95	65	13.34
maintainability_index (%)	23.46	100	82.958

Частотное распределение значений по метрике сложности по Холстеду

Метрика	Диапазон значений									
	0–7	7–14	14–21	21–28	28–35	35–42	42–49	49–56	56–63	63–70
halstead difficulty	36	110	34	21	19	1	0	1	1	2

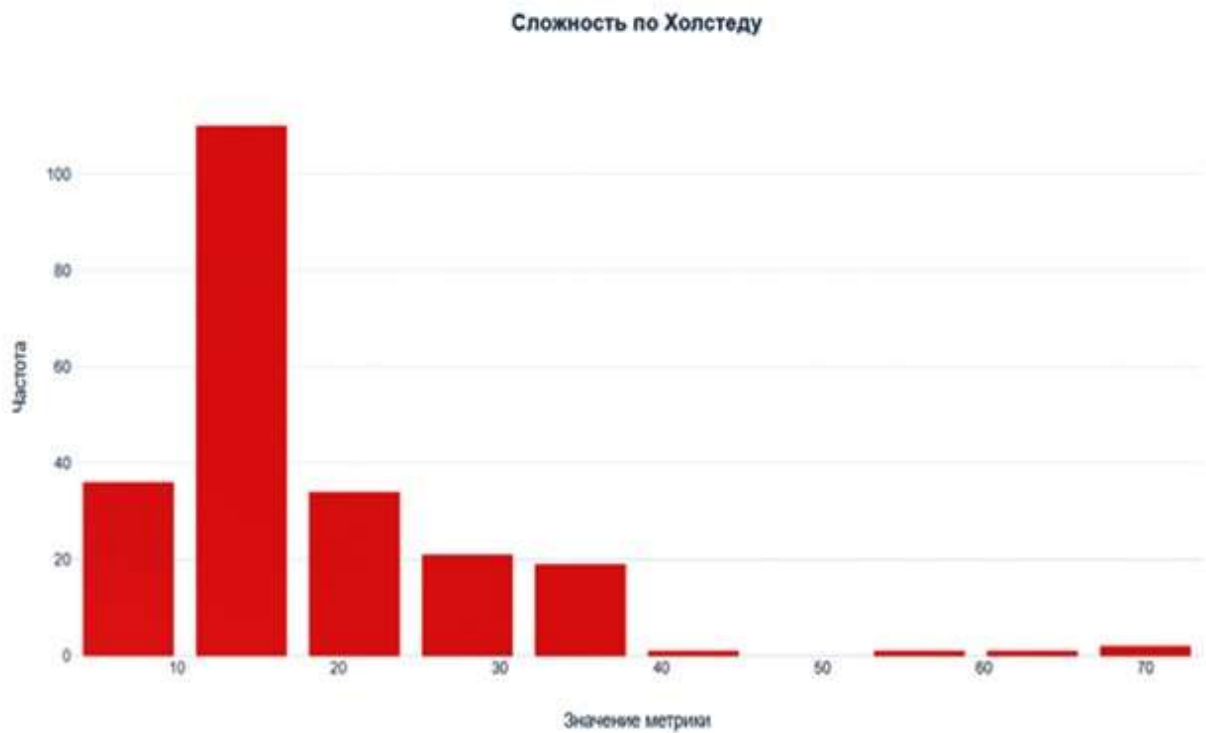


Рисунок 11. График распределения значений по метрике сложности по Холстеду

Частотное распределение значений по метрике индекса поддерживаемости кода

Метрика	Диапазон значений									
	0–10	10–20	20–30	30–40	40–50	50–60	60–70	70–80	80–90	90–100
maintainability_index (%)	0	0	3	2	6	5	12	56	78	17

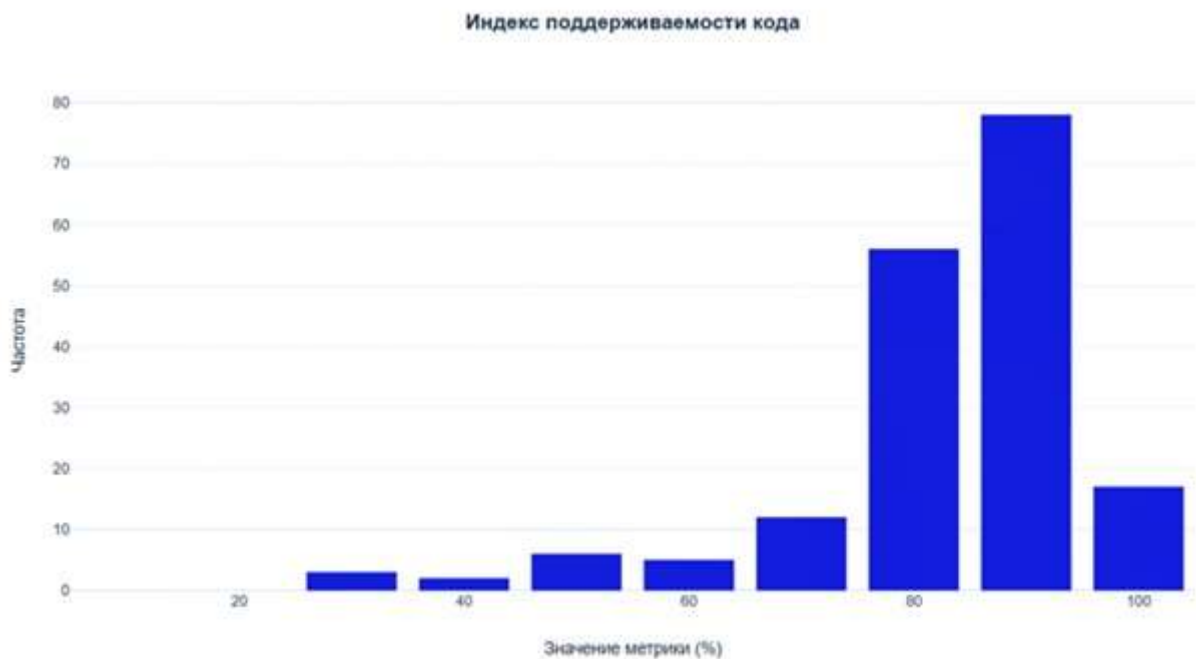


Рисунок 12. График распределения значений по метрике индекса поддерживаемости кода

После этого для каждой метрики были взяты средние значения частотных диапазонов по каждому заданию и разделены на 5 категорий от худшего показателя к лучшему. Результаты распределения метрик по категориям указаны в таблице 15.

Распределение метрик по категориям

Категория	Метрики и диапазоны			
	Цикломатическая сложность	Сложность по Холстеду	Индекс поддерживаемости кода (%)	% комментариев в коде
Е	>20	>80	0–30	0–3
D	15–20	60–80	30–45	3–5
C	10–15	45–60	45–60	5–10
B	5–10	25–45	60–80	10–15
A	<5	0–25	80–100	>15

С помощью библиотеки `multimetric` и написанного `bash`-скрипта (см. приложение А) для получения результатов в `JSON`-формате можно получить все описанные ранее метрики как по каждому файлу, так и со статистическими показателями (среднее, медиана, минимум, максимум и т.д.).

На рисунке 13 показан расчет метрик со статистикой по тестовой выборке.



Рисунок 13. Статистические метрики по набору файлов из обучающей выборки

На примере можно заметить, что статистические значения в первую очередь берутся исходя из количества строк. Для максимального и медианного значения индекс поддержки программного кода (см. параметр `maintainability_index`) равен 100%, для минимального – 70.4%. Это означает, что пример с минимальным количеством строк содержит слишком длинные строки в среднем, а также циклы в этих строках (пример – тернарный оператор, который можно записать в одну строку, но при этом усложнить читаемость кода в случае их множественного количества), что за собой влечёт худшую степень поддержки программного кода.

Для демонстрации полученных результатов метрик оценки программного кода была взята лабораторная работа, в которой требовалось реализовать алгоритм быстрой сортировки. На рисунках 14–17 показаны значения метрик оценки цикломатической сложности программы, процента комментариев кода, сложности программы по Холстеду, а также общая оценка качества по рейтингу ТЮВЕ (по оси X – `id` студента, по оси Y – значение метрики). Красной линией обозначены пороговые значения метрик, определенные на основании исходных значений выборки. Для цикломатической сложности и сложности по Холстеду – чем больше значение метрики, тем хуже качество кода; для процента комментариев: если меньше 5% комментариев, то качество считается хуже.



Рисунок 14. Цикломатическая сложность кода

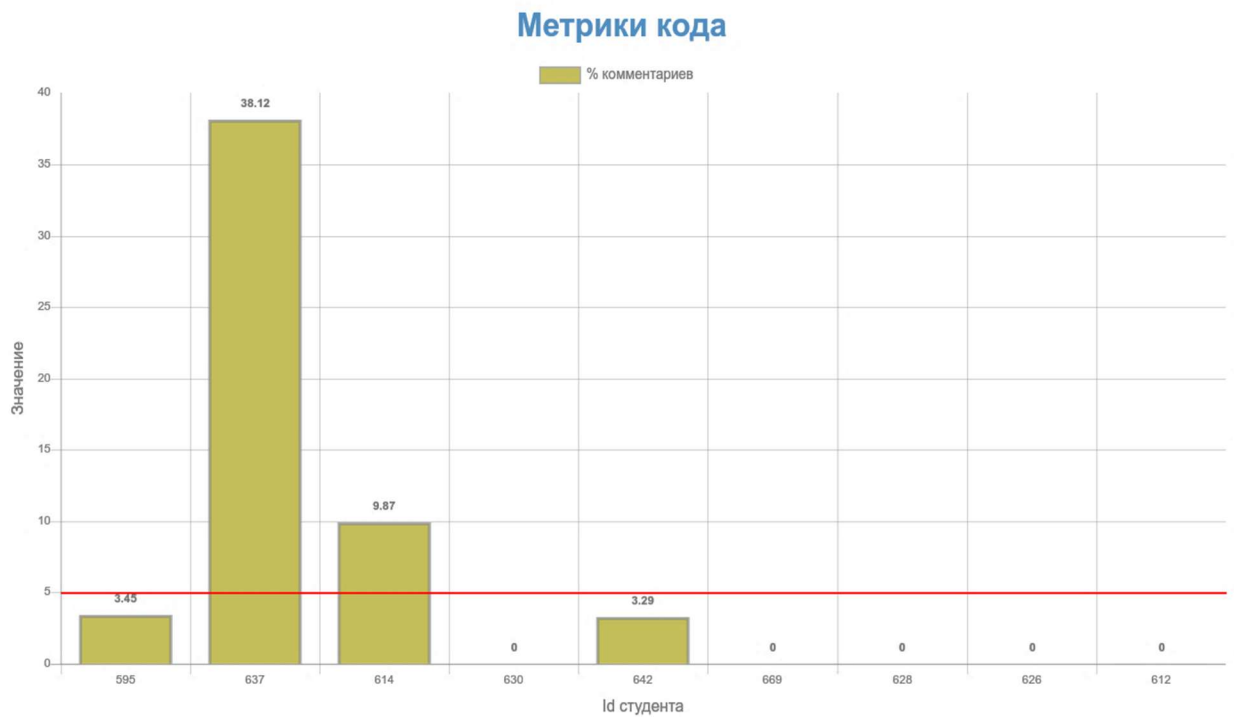


Рисунок 15. Процент комментариев в коде

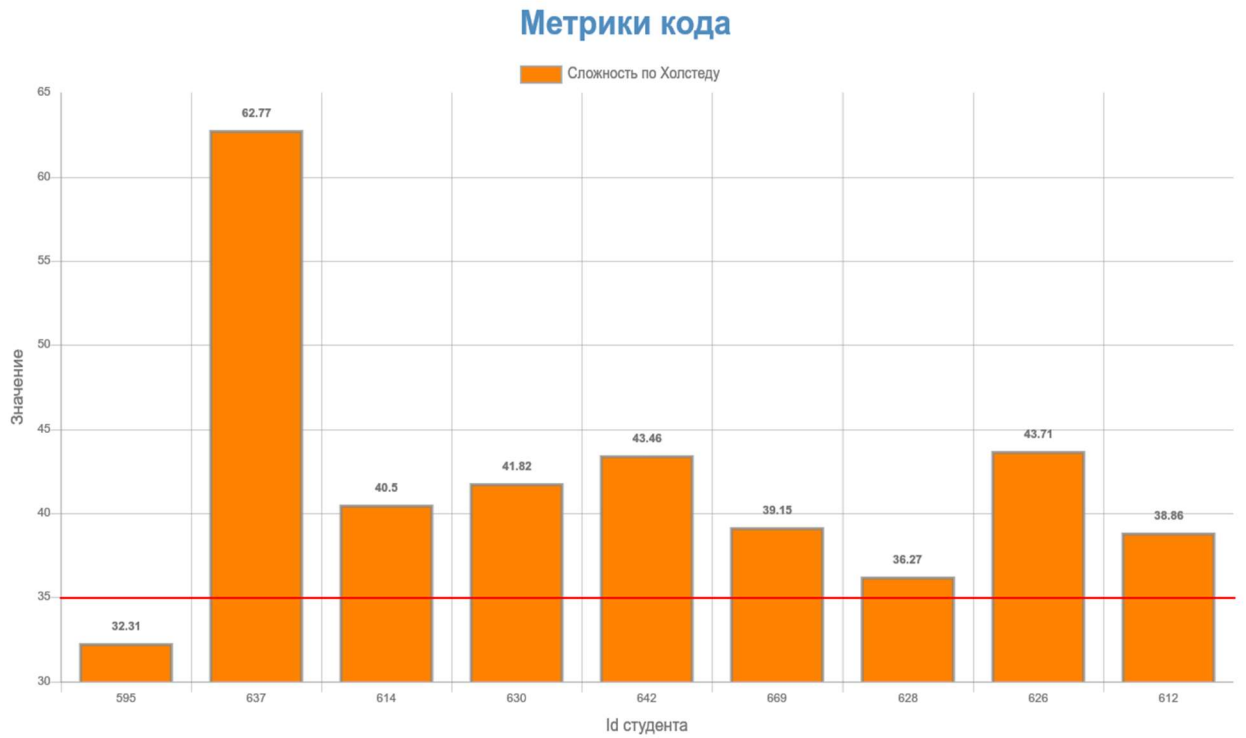


Рисунок 16. Сложность кода по Холстеду

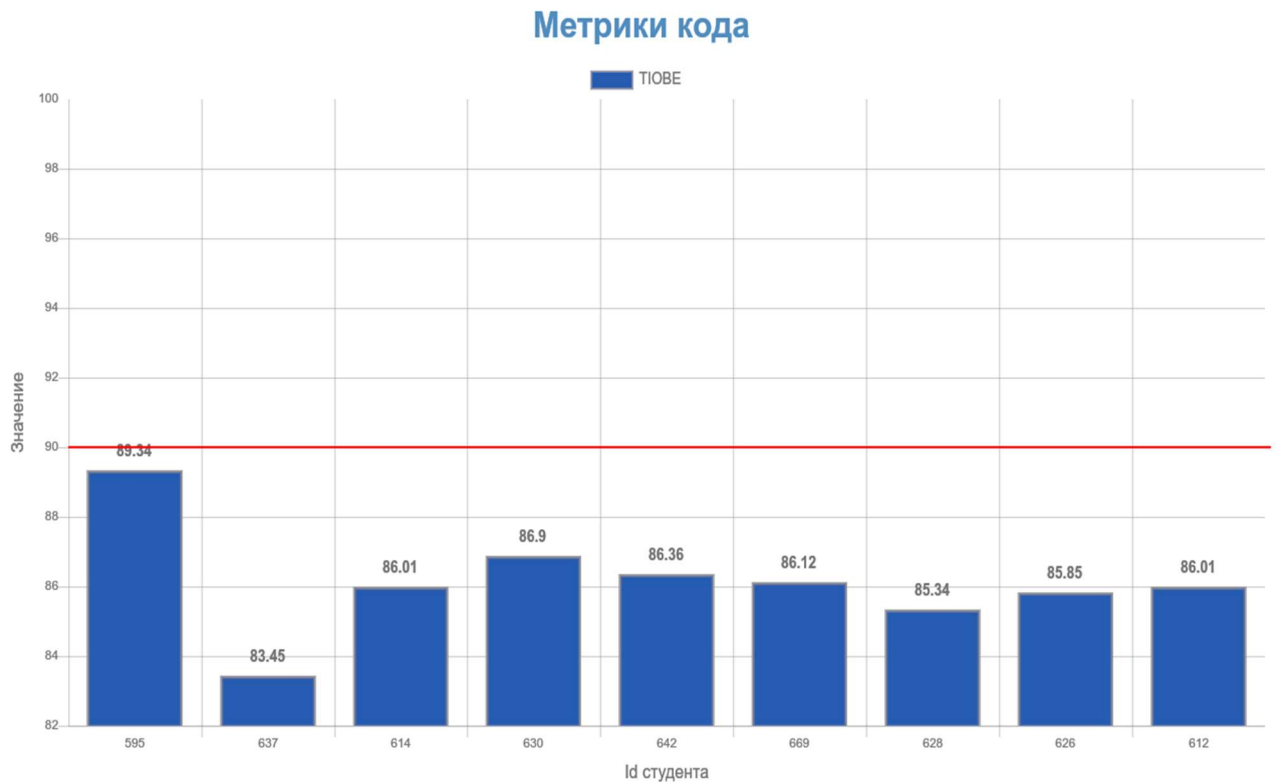


Рисунок 17. Рейтинг TIOBE

Данный пример является показательным тем, что студент с идентификатором 637, несмотря на большой процент комментариев в коде,

имеет показатели цикломатической сложности и сложности по Холстеду значительно выше рекомендуемых значений, что указывает на избыточность кода и его возможную низкую эффективность, в то время как у студента с идентификатором 595 вышеуказанные оценки сложности кода имеют минимальные значения.

Для приведения метрик к единому формату применена линейная нормировка по следующему принципу:

Формула для метрик с градацией «больше–лучше»:

$$\hat{x} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (7)$$

Формула для метрик с градацией «меньше–лучше»:

$$\hat{x} = 1 - \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (8)$$

Общая оценка определяется по формуле $score = A * CC + B * HC + C * HV + D * LOC + E * MI + F * CP$ (9), где:

- CC – цикломатическая сложность (A = 0.25).
- HC – сложность по Холстеду (B = 0.2).
- HV – объем по Холстеду (C=0.2).
- LOC – число строк кода (D=0.15).
- MI – индекс поддерживаемости кода (E=0.15).
- CP – доля комментариев в коде (F=0.05).

и значение score находится в диапазоне от 0 до 1.

В таблице 16 показано соответствие общей оценки к категории и её интерпретация.

Таблица 16

Определение категории по оценке и интерпретация

Оценка	Категория	Комментарий
0–0.2	E	Код нуждается в полной переработке
0.2–0.4	D	Код имеет недостатки и требует доработки модулей

0.4–0.6	C	Код нуждается в рефакторинге
0.6–0.8	B	Есть незначительные недочеты в коде
0.8–1	A	Код оптимален по объему и эффективен в использовании

Для общего сравнения кодов студентов используется отображение на лепестковой диаграмме (см. рисунок 18). Качество кода на ней интерпретируется следующим образом: чем больше площадь графика, тем выше значение по категории.

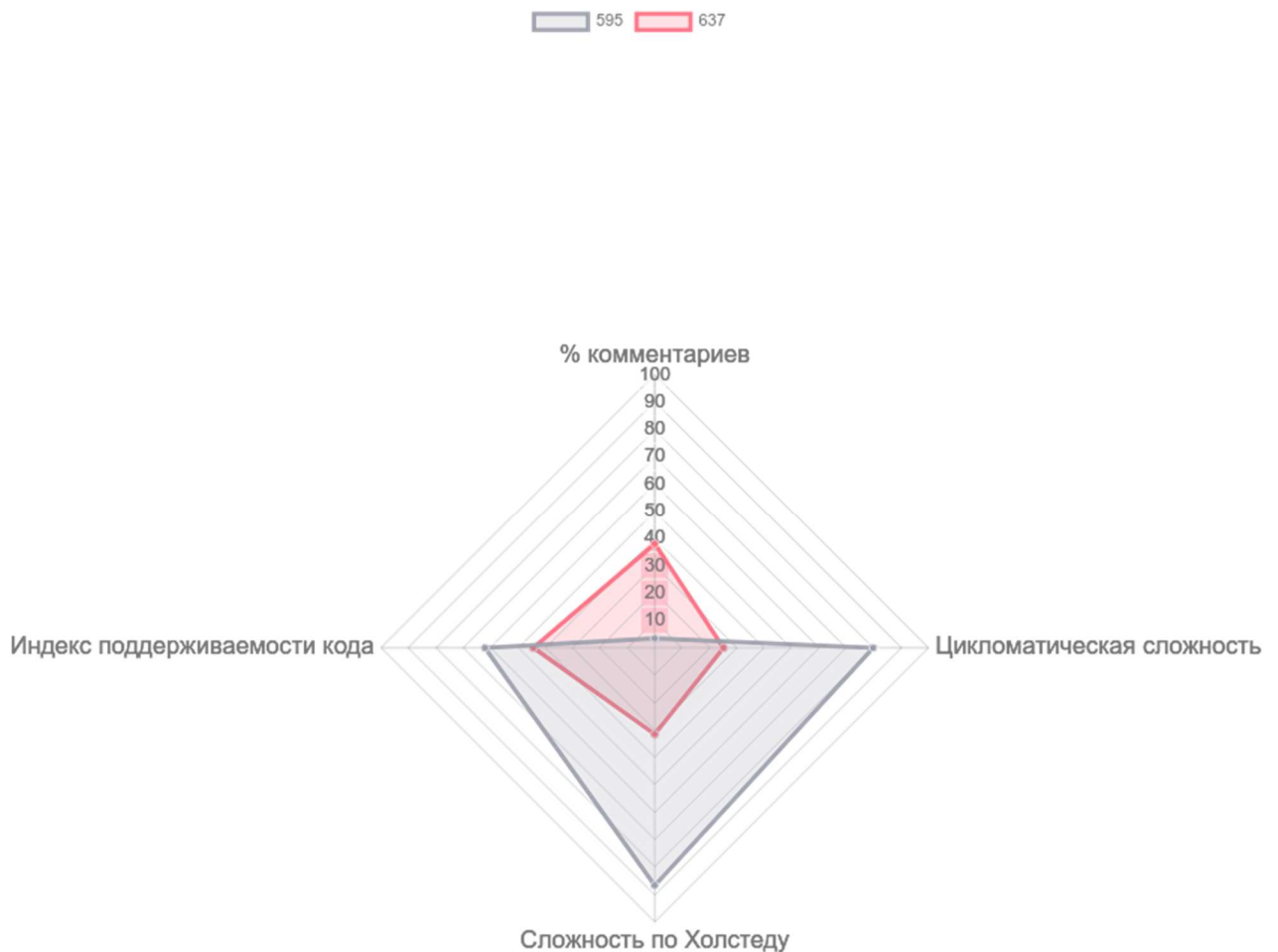


Рисунок 18. Лепестковая диаграмма сравнения кодов студентов

В листингах 1 и 2 показан исходный код двух программ для их сравнения (персональные данные убраны).

Листинг 1. Исходный код студента с идентификатором 637

```
#include <iostream> /******* ***, *****
#include <ctime> //Таблица "Быстрая сортировка"
#include <iomanip>

using namespace std;

int quickcomparison(int*, int, int); //выполняет быструю сортировку +
считает сравнения
int quickswap(int*, int, int); //выполняет быструю сортировку + считает
обмены

int main()
{
    setlocale(LC_ALL, "Russian");
    int const A = 50000;
    int const B = 100000;
    int const C = 1000000;
    int N[3];
    int RES[2][3];
    int i, j, first, xlast, ylast, zlast;
    int*x = new int[A];
    int*y = new int[B];
    int*z = new int[C];
    srand(time(0));
    for (i = 0; i < A; i++) //заполнение массива x случайными числами
    {
        x[i] = rand() % 10;
    }
    srand(time(0));
    for (i = 0; i < B; i++) //заполнение массива y случайными числами
    {
        y[i] = rand() % 10;
    }
}
```

```
srand(time(0));
for (i = 0; i < C; i++) //заполнение массива z случайными числами
{
    z[i] = rand() % 10;
}
first = 0;
xlast = A - 1;
ylast = B - 1;
zlast = C - 1;
N[0] = A;
N[1] = B;
N[2] = C;
RES[0][0] = quickcomparison(x, first, xlast);
RES[1][0] = quickswap(x, first, xlast);
RES[0][1] = quickcomparison(y, first, ylast);
RES[1][1] = quickswap(y, first, ylast);
RES[0][2] = quickcomparison(z, first, zlast);
RES[1][2] = quickswap(z, first, zlast);
cout << setw(19);
for (j = 0; j <= 2; j++) //вывод количества элементов
{
    cout << N[j] << setw(11);
}
cout << endl;
for (i = 0; i <= 1; i++) //вывод результатов
{
    switch (i)
    {
        case 0: cout << "Сравнения: "; break;
    }
    switch (i)
    {
        case 1: cout << "Обмены: "; break;
    }
}
```

```

for (j = 0; j <= 2; j++)
{
    cout << setw(10) << RES[i][j];
}
if ((i + 1) % 2 == 0) //массив выводится построчно
{
    cout << "\n" << "\n";
}
else
{
    cout << "\n";
}
}
}

int quickcomparison(int*mas, int first, int last) //принимает массив,
индексы первого и последнего элементов этого массива
{
    int mid, tmp;
    int comparison = 0;
    int f = first, l = last;
    mid = mas[(f + 1) / 2]; //определяем значение опорного элемента
do
{
    while (mas[f] < mid) //ищем индекс первого встречающегося
"проблемного" элемента слева
    {
        f++;
        comparison++;
    }
    while (mas[l] > mid) //ищем индекс первого встречающегося
"проблемного" элемента справа
    {
        l--;
    }
}
}

```

```

        comparison++;
    }
    if (f <= l) //меняем местами элементы
    {
        tmp = mas[f];
        mas[f] = mas[l];
        mas[l] = tmp;
        f++;
        l--;
    }
} while (f < l); //цикл выполняется до тех пор, пока f и l не
встретятся
    if (first < l) //если образовавшийся подмассив содержит в себе
более одного элемента, то...
    {
        comparison += quickcomparison(mas, first, l); //... вызов
функции в функции
    }
    if (f < last) //если образовавшийся подмассив содержит в себе более
одного элемента, то...
    {
        comparison += quickcomparison(mas, f, last); //вызов функции в
функции
    }
    return comparison;
}

int quickswap(int*mas, int first, int last) //принимает массив, индексы
первого и последнего элементов этого массива
{
    int mid, tmp;
    int swap = 0;
    int f = first, l = last;
    mid = mas[(f + l) / 2]; //определяем значение опорного элемента

```



```

do
{
    while (mas[f] < mid) //ищем индекс первого встречающегося
"проблемного" элемента слева
    {
        f++;
    }
    while (mas[l] > mid) //ищем индекс первого встречающегося
"проблемного" элемента справа
    {
        l--;
    }
    if (f <= l) //меняем местами элементы
    {
        tmp = mas[f];
        mas[f] = mas[l];
        mas[l] = tmp;
        f++;
        l--;
        swap++;
    }
} while (f < l); //цикл выполняется до тех пор, пока f и l не
встретятся
    if (first < l) //если образовавшийся подмассив содержит в себе
более одного элемента, то...
    {
        swap += quickswap(mas, first, l); //... вызов функции в
функции
    }
    if (f < last) //если образовавшийся подмассив содержит в себе более
одного элемента, то...
    {
        swap += quickswap(mas, f, last); //вызов функции в функции
    }

```

```

    return swap;
}
;

```

Листинг 2. Исходный код студента с идентификатором 595

```

#include <iostream>
#include <conio.h>
#include <time.h>
using namespace std;
void quickSort(int arr[], int left, int right)
{
    int i = left, j = right;
    int tmp;
    int Sr = 0, Ob = 0;
    int pivot = arr[(left + right) / 2];
    /* partition */
    while (i <= j)
    {
        Sr++;
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (i <= j)
        {
            Ob++;
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
        cout << "кол-во сравнений = " << Sr << endl;
        cout << "кол-во обменов = " << Ob << endl;
    };
}

```

```
/* recursion */
if (left < j)
{
    quickSort(arr, left, j);
}
if (i < right)
{
    quickSort(arr, i, right);
}
}
int main() {
    setlocale(LC_CTYPE, "Russian");
    int num;
    cout << "Колво элементов: ";
    cin >> num;

    int* mass = new int[num];
    srand((unsigned)time(NULL));
    for (int i = 0; i < num; i++)
    {
        mass[i] = rand() % 20 + 100;
        cout << mass[i] << " ";
    }
    cout << endl;
    cout << endl;
    cout << "Quicksorted array:" << endl;
    clock_t t0 = clock();
    quickSort(mass, 0, num - 1); // функция квиксорта.
    for (int i = 0; i < num; i++)
    {
        cout << mass[i] << "\t";
    }
    clock_t t1 = clock();
    cout << endl;
```

```

cout << "time: " << (double)(t1 - t0) / CLOCKS_PER_SEC << endl;
return 0;
};

```

После применения и проверки решений был написан API для работы с операциями по программным кодам (см. приложение Б и рисунок 19), в частности, операцией добавления программного кода студента в хранилище HBase с привязкой по идентификатору в базе данных PostgreSQL для дальнейшего обращения к записи из хранилища. Далее был создан веб-модуль, состоящий из ASP.NET и Python для бэкенда и Jinja2 для фронтенда. В приложении В и на рисунке 20 показана реализация модуля в веб-интерфейсе по созданию групп, для которых в дальнейшем будет проводиться динамика работ студентов.

Students	
GET	/StudentsGroups/Students
StudentsCodes	
GET	/StudentsCodes/StudentsCodes/{student_id}/{task_id}
POST	/StudentsCodes/StudentsCodes/{student_id}/{task_id}/{filePath}
PUT	/StudentsCodes/StudentsCodes/{student_id}/{task_id}/{filePath}
PUT	/StudentsCodes/StudentsCodes/{hbase_id}/{filePath}
DELETE	/StudentsCodes/StudentsCodes/{id}
StudentsGroups	
GET	/StudentsGroups/StudentsGroups/{id}
DELETE	/StudentsGroups/StudentsGroups/{id}
PUT	/StudentsGroups/StudentsGroups
POST	/StudentsGroups/StudentsGroups

Рисунок 19. API для работы с группами и кодами студентов

Список групп

Группа по алфавиту ▾
- Удалить группу

Группа по алфавиту

Альфа

- Убрать подгруппу

Выберите студента ▾

Список выбранных студентов

Абакулов Бекзод Санъатбекович

Абиева Севиндж Абизар

Абросимов Максим Евгеньевич

Бета

- Убрать подгруппу

Выберите студента ▾

Список выбранных студентов

Бондаренко Екатерина Андреевна

Биримжанов Дамир Куаншевич

Белозеров Николай Викторович

Тета

- Убрать подгруппу

Выберите студента ▾

Список выбранных студентов

Торопыгин Антон Юрьевич

Ташбулатов Ренат Маратович

Рисунок 20. Модуль формирования групп

ЗАКЛЮЧЕНИЕ

В результате работы были реализованы следующие задачи:

- изучены и разобраны различные признаки, а также их назначение и возможное применение при статическом анализе кода для его оценки;
- проведен анализ имеющихся признаков на основании большого набора данных в виде исходного кода из открытых источников для определения пороговых значений и их категоризации по каждому из признаков;
- создан API для обработки программных кодов, а также создан веб-модуль, состоящий из модуля для формирования групп преподавателем, а также из модуля визуализации по каждому из признаков для преподавателей с целью оценки программных кодов студентов.

Реализованный механизм анализа программного кода позволяет преподавателям более объективно оценивать работы студентов, минимизировав субъективную оценку, а также самим студентам рассмотреть динамику своих работ и выявить недочеты, которые необходимо разобрать и проработать для их исправления.

В дальнейшем планируется реализация варьируемой оценки кода в зависимости от приоритета признаков с реализацией через набор параметров с возможностью их ручной регулировки.

СПИСОК ЛИТЕРАТУРЫ

1. Cornett, S. Code Coverage Analysis [Электронный ресурс]. URL: <http://www.bullseye.com/coverage.html> (дата обращения: 15.04.2021).
2. Cppcheck – A tool for static C/C++ code analysis [Электронный ресурс]. URL: <http://cppcheck.sourceforge.net/> (дата обращения: 03.04.2021).
3. E. Enslin, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis // In Proceedings of the 6th International Working Conference on Mining Software Repositories, pp. 71–80, 2009.
4. Enas Alikhashashneh, Rajeev Raje, James Hill. Using Software Engineering Metrics to Evaluate the Quality of Static Code Analysis Tools. [Электронный ресурс]. URL: <https://ieeexplore.ieee.org/document/8367641> (дата обращения: 14.06.2021).
5. ISO/IEC 25010:2011. Systems and software engineering // Systems and software Quality Requirements and Evaluation (SQuARE) — System and software quality models [Электронный ресурс]. URL: <https://www.iso.org/standard/35733.html> (дата обращения: 15.04.2021).
6. Joseph Albahari, Ben Albahari. C# 7.0 in a Nutshell, 7th edition // O'Reilly, Chapter 27. The Roslyn Compiler, pp. 1007–1020, 2018.
7. multimetric - Calculate code metrics in various languages. [Электронный ресурс]. URL: <https://github.com/priv-kweihmann/multimetric> (дата обращения: 20.04.2021).
8. Paul Jansen. The TIOBE Quality Indicator [Электронный ресурс]. URL: https://www.tiobe.com/files/TIOBEQualityIndicator_v4_8.pdf (дата обращения: 15.04.2021).
9. Pintér Ádám, Szénási Sándor. Preprocessed C# Source Codes for Machine Learning [Электронный ресурс]. URL: https://figshare.com/articles/dataset/Preprocessed_C_Source_Codes_for_Machine_Learning/8428229/1 (дата обращения: 23.05.2021).

10. Static analysis (static code analysis). [Электронный ресурс]. URL: <https://whatis.techtarget.com/definition/static-analysis-static-code-analysis> (дата обращения: 15.06.2021).
11. Sudipta Mukherjee. Source Code Analytics With Roslyn and JavaScript Data Visualization // Apress, pp. 36–52, 2016.
12. T.J. McCabe. A complexity measure // IEEE Transactions on Software Engineering, vol. SE–2, no. 4, pp. 308–320, 1976.
13. The Unicode Standard // Byte Order Mark (BOM): U+FEFF, chapter 16.8, pp. 550–553, 2007 [Электронный ресурс]. URL: <https://www.unicode.org/versions/Unicode5.0.0/ch16.pdf#G25817> (дата обращения: 25.05.2021).
14. Wiley–IEEE Computer Society, New York, USA. Software Metrics and Software Metrology // Chapter 7. Halstead Metrics, Analysis of their Design, pp. 145–159, 2010.
15. Введение в Roslyn. Использование для разработки инструментов статического анализа. [Электронный ресурс]. URL: <https://habr.com/ru/company/pvs-studio/blog/301204/> (дата обращения: 11.06.2021).
16. Документация по анализатору кода PVS–Studio [Электронный ресурс]. URL: <https://pvs-studio.com/ru/docs/manual/full/> (дата обращения: 03.04.2021).
17. Документация по использованию доступных лексеров в библиотеке pygments // Available lexers — Pygments [Электронный ресурс]. URL: <https://pygments.org/docs/lexers/> (дата обращения: 29.04.2021).
18. Использование статического и динамического анализа для повышения качества продукции и эффективности разработки. [Электронный ресурс]. URL: <https://www.swd.ru/print.php3?pid=828> (дата обращения: 15.06.2021).
19. Ошибки, обнаруженные в Open Source проектах разработчиками PVS–Studio с помощью статического анализа [Электронный ресурс]. URL: <https://www.viva64.com/ru/examples/> (дата обращения: 03.04.2021).

20. Программный код и его метрики (блог компании Intel). [Электронный ресурс]. URL: <https://habr.com/ru/company/intel/blog/106082/> (дата обращения: 30.03.2021).

Приложение А. bash–скрипт для работы библиотеки multimetric

```
#!/bin/bash
MY_PATH=`dirname \"$0\"`
if [ -z \"$MY_PATH\" ] ; then
    exit 1
fi

echo $MY_PATH

FILENAMES=$(find ${MY_PATH} -maxdepth 1 -type f -not -path '*/\.*' -not -name \"*.sh\" |
sed 's/^\.\./g' | sort | head -n 1000)
echo $(multimetric ${FILENAMES}) > calced.json
read -p \"Press enter to continue\"
```

Приложение Б. API для работы с программным кодом

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Hbase.Thrift;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Npgsql;
using Thrift.Protocols;
using Thrift.Transports.Client;

// For more information on enabling Web API for empty projects, visit
https://go.microsoft.com/fwlink/?LinkID=397860

namespace DSSharp.Controllers
{
    [Route(\"StudentsCodes/[controller]\")]
    [ApiController]
    public class StudentsCodesController : ControllerBase
    {
        private readonly IConfiguration _configuration;
        private readonly string connString;
        private readonly string hBaseHost;
        private readonly int hBasePort;
        public StudentsCodesController(IConfiguration cfg)
        {
```

```

_configuration = cfg;
connString = cfg.GetSection("ConnectionStrings")
                .GetSection("PostgreSQLConnection").Value;
hBaseHost = cfg.GetSection("ConnectionStrings")
                .GetSection("HBaseConnection")
                .GetSection("ServerHostName").Value;
hBasePort = Convert.ToInt32(
                cfg.GetSection("ConnectionStrings")
                .GetSection("HBaseConnection")
                .GetSection("Port").Value);
}

[HttpGet("{student_id}/{task_id}")]
public async Task<string> Get(int student_id, string task_id)
{
    Hbase.Thrift.Hbase.Client _hBase;
    string serverHostName = hBaseHost;
    int port = hBasePort;
    var ip = System.Net.IPAddress.Parse(serverHostName);
    var transport = new TBufferedClientTransport(new TSocketClientTransport(ip,
port));

    var protocol = new TBinaryProtocol(transport);
    _hBase = new Hbase.Thrift.Hbase.Client(protocol);
    await transport.OpenAsync();

    await using (Database.StudentsAndGroupsContext db = new
Database.StudentsAndGroupsContext())
    {
        var codeMeta = db.CodeMetas.First(x => x.StudentId == student_id &&
                x.TaskId == task_id);

        var guid = codeMeta.HBaseGuid;
        try
        {
            byte[] tableName = "codes".GetUTF8Bytes();
            byte[] rowGuid = guid.GetUTF8Bytes();
            var row = await _hBase.getRowAsync(tableName, rowGuid, null,

System.Threading.CancellationToken.None);

            var code =
row.First().Columns.Values.First().Value.GetUTF8String();

```

```

        return code;
    }
    catch (Exception ex)
    {
        return string.Empty;
    }
}

// POST api/<StudentsCodesController>
[HttpPost("{student_id}/{task_id}/{filePath}")]
public async Task Post(int student_id, string task_id, string filePath)
{
    Hbase.Thrift.Hbase.Client _hBase;
    string serverHostName = hBaseHost;
    int port = hBasePort;

    string code = System.IO.File.ReadAllText(filePath);
    var guid = Guid.NewGuid().ToString().Replace("-", "");
    await using (Database.StudentsAndGroupsContext db = new
Database.StudentsAndGroupsContext())
    {
        await db.CodeMetas.AddAsync(new Database.CodeMetas
        {
            StudentId = student_id,
            HBaseGuid = guid,
            TaskId = task_id
        });
        await db.SaveChangesAsync();
    }

    var ip = System.Net.IPAddress.Parse(serverHostName);
    var transport = new TBufferedClientTransport(new TSocketClientTransport(ip,
port));

    var protocol = new TBinaryProtocol(transport);
    _hBase = new Hbase.Thrift.Hbase.Client(protocol);

    try
    {
        await transport.OpenAsync();
    }
}

```

```

byte[] tableName = "codes".GetUTF8Bytes();
byte[] row = guid.GetUTF8Bytes();
Mutation _mutation = new Mutation();
_mutation.IsDelete = false;
_mutation.Column = "code:text".GetUTF8Bytes();
_mutation.Value = code.GetUTF8Bytes();
await _hBase.mutateRowAsync(tableName, row,
                             new List<Mutation> { _mutation },
                             null,
                             System.Threading.CancellationToken.None);
}
catch (Exception e)
{
}
}
}
[HttpPut("{student_id}/{task_id}/{filePath}")]
public async Task Put(int student_id, string task_id, string filePath)
{
    Hbase.Thrift.Hbase.Client _hBase;
    string serverHostName = hBaseHost;
    int port = hBasePort;

    string code = System.IO.File.ReadAllText(filePath);
    Database.CodeMetas codeMeta;
    await using (Database.StudentsAndGroupsContext db = new
Database.StudentsAndGroupsContext())
    {
        codeMeta = db.CodeMetas.First(x => x.StudentId == student_id &&
x.TaskId == task_id);
    }

    var ip = System.Net.IPAddress.Parse(serverHostName);
    var transport = new TBufferedClientTransport(new TSocketClientTransport(ip,
port));

    var protocol = new TBinaryProtocol(transport);
    _hBase = new Hbase.Thrift.Hbase.Client(protocol);

    try
    {

```

```

        await transport.OpenAsync();
        byte[] tableName = "codes".GetUTF8Bytes();
        byte[] row = codeMeta.HBaseGuid.GetUTF8Bytes();
        Mutation _mutation = new Mutation();
        _mutation.IsDelete = false;
        _mutation.Column = "code:text".GetUTF8Bytes();
        _mutation.Value = code.GetUTF8Bytes();
        await _hBase.mutateRowAsync(tableName, row,
                                    new List<Mutation> { _mutation },
                                    null,
                                    System.Threading.CancellationToken.None);
    }
    catch (Exception e)
    {
    }
}

[HttpPut("{hbase_id}/{filePath}")]
public async Task Put(string hbase_id, string filePath)
{
    var code = System.IO.File.ReadAllText(filePath);
    Hbase.Thrift.Hbase.Client _hBase;
    string serverHostName = hBaseHost;
    int port = hBasePort;

    var ip = System.Net.IPAddress.Parse(serverHostName);
    var transport = new TBufferedClientTransport(new TSocketClientTransport(ip,
port));
    var protocol = new TBinaryProtocol(transport);
    _hBase = new Hbase.Thrift.Hbase.Client(protocol);

    try
    {
        await transport.OpenAsync();
        byte[] tableName = "codes".GetUTF8Bytes();
        byte[] row = hbase_id.GetUTF8Bytes();
        Mutation _mutation = new Mutation();
        _mutation.IsDelete = false;
        _mutation.Column = "code:text".GetUTF8Bytes();
    }
}

```

```

        _mutation.Value = code.GetUTF8Bytes();
        await _hBase.mutateRowAsync(tableName, row,
                                    new List<Mutation> { _mutation },
                                    null,
                                    System.Threading.CancellationToken.None);
    }
    catch (Exception e)
    {
    }
}

// DELETE api/<StudentsCodesController>/5
[HttpDelete("{id}")]
public void Delete(int id)
{
}
}
}

```

Приложение В. Реализация создания групп

```

<template>
  <div id="app">
    <h1 class="group-1st-hdr">Список групп</h1>
    <GroupList v-bind:groups="groups" v-bind:all_students="all_students"
              @on-change="onChange"
              @on-delete="delGroup">

    </GroupList>
  </div>
</template>

<script>
import GroupList from './components/GroupList.vue'
//import groups_json from './json_src/groups_json.json'
//import all_students_json from './json_src/all_students_json.json'
export default {
  name: 'App',
  data() {
    return {
      groups: [],

```

```

    all_students: []
  }
},
components: {
  GroupList
},
mounted() {
  fetch('https://localhost:44384/StudentsGroups/Students')
  .then(response => response.json())
  .then(json => this.all_students = json)
  fetch('https://localhost:44384/StudentsGroups/StudentsGroups/5')
  .then(response => response.json())
  .then(json => this.groups = json)
  .then(j => this.editJson(j))
},
methods: {
  getObjectId() {
    var timestamp = (new Date().getTime() / 1000 | 0).toString(16);
    return timestamp + 'xxxxxxxxxxxxxxxx'.replace(/[x]/g, function() {
      return (Math.random() * 16 | 0).toString(16);
    }).toLowerCase();
  },
  onChange(event) {
    var index = this.groups.length + 1
    var objId = this.getObjectId()
    if(event.target.value === "add_new_group") {
      var teacher_id = 5
      var title = "Грpynna " + index
      var json = {"id": objId, "title": title, "isVisible": true}
      this.groups.push(json)
      this.groups.slice(-1)[0].isVisible = true
      fetch('https://localhost:44384/StudentsGroups/StudentsGroups?teacher_id='+
teacher_id +
      '&groupName=' + title +
      '&groupId=' + objId +
      '&json=' + JSON.stringify(json),
      {
        method: 'POST'
      }
    )
  }
}
}

```



```

    }
    this.groups.forEach(
      function(item) {
        //if(event.target.value === "add_new_group")
        // item.isVisible = (Number(item.id) === Number(index));
        //else {
          item.isVisible = (item.id === event.target.value);
        //}
      }
    );
  },
  delGroup(id) {
    this.groups = this.groups.filter(x => x.id !== id)
    fetch('https://localhost:44384/StudentsGroups/StudentsGroups/' + id, {
      method: "DELETE"
    })
  },
  editJson(j) {
    var arr = []
    j.forEach(
      function(item) {
        var group = JSON.parse(item.group)
        group.isVisible = false
        arr.push(group)
      }
    );
    this.groups = arr;
  }
}
</script>

<style>
.group-1st-hdr {
  font-family: Calibri;
  text-align: left;
}
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;

```

```

-moz-osx-font-smoothing: grayscale;
text-align: left;
color: #2c3e50;
margin-top: 60px;
}
</style>
<template>
  <div v-if="group.isVisible">
    <button class="del_group" @click="$emit('on-delete', group.id)">- Удалить
    грруппу</button>
    <p><input type="text" class="group_title" :value="group.title" v-
    on:input="group.title = $event.target.value"/></p>
    <div class="subgroups-div" v-for="subgroup in group.subgroups" v-
    bind:key="subgroup.id">
      <p><input type="text" :value="subgroup.title" v-
      on:input="subgroup.title = $event.target.value"/><button @click="delSubgroup(subgroup)"
      class="remove_subgroup">- Убрать подгруппу</button></p>
      <select :id="subgroup.id" :name="subgroup.title"
      @change="onChange($event, subgroup.id)">
        <option selected disabled>Выберите студента</option>
        <option v-for="(stud, index) in all_students" v-
        bind:value="stud.id" :key="index">{{stud.surname + " " + stud.name + " " +
        stud.patronymic}}</option>
      </select>
      <h4 v-if="subgroup.students.length > 0">Список выбранных
      студентов</h4>
      <ul class="selector">
        <li v-for="(student, index) in subgroup.students"
        :class="student.id" :key="index">{{student.user}}<button class="del_stud_btn"
        @click="delStudent(subgroup, index)">Удалить</button></li>
      </ul>
    </div>
    <div v-if="group.subgroups === undefined">
      <p>Подгрупп нет</p>
    </div>
    <button @click="addSubgroup">+ Добавить новую подгруппу</button>
    <br/>
    <button class="save_group" @click="saveGroup(group)" v-if="group.subgroups
    != undefined">Обновить грруппу</button>

```

```

        <button class="draw_graphs" @click.prevent="drawGroups(group)">Отрисовать
график</button>
    <p></p>
    <div class="subgroup_graphs" v-for="subgroup in group.subgroups" v-
bind:key="subgroup.name">
        <h4>{{subgroup.title}}</h4>
        <GroupChart :labels="labels" :datasets="datasets"
ref="vuechart"></GroupChart>
    </div>
</div>
</template>

<script>
import GroupChart from '@/components/GroupChart'
export default {
  name: 'group-view',
  props: {
    group: {
      type: Object,
      required: true
    },
    all_students: {
      type: Array,
      required: true
    }
  },
  components: {
    GroupChart
  },
  methods: {
    getObjectId() {
      var timestamp = (new Date().getTime() / 1000 | 0).toString(16);
      return timestamp + 'xxxxxxxxxxxxxxxx'.replace(/[x]/g, function() {
        return (Math.random() * 16 | 0).toString(16);
      }).toLowerCase();
    },
    onChange(event, id) {
      console.log(id)
      var e = document.getElementById(id);
      var strUser = e.options[e.selectedIndex].innerText;

```

```

    var studs = this.group.subgroups.find(x => x.id === id).students;
    if(studs.find(x => x == event.target.value) === undefined)
        studs.push({"id": event.target.value, "user" : strUser })
    },
    addSubgroup() {
        var objectId = this.getObjectId()
        if (this.group.subgroups === undefined)
            this.$set(this.group, 'subgroups', [])
        var index = this.group.subgroups.length + 1
        this.group.subgroups.push({"id":objectId, "title":"Подгруппа " +
index,"students":[]});
    },
    delStudent(subgroup, index) {
        subgroup.students.splice(index, 1)
    },
    delSubgroup(subgroup) {
        this.group.subgroups = this.group.subgroups.filter(x => x.id !=
subgroup.id)
    },
    saveGroup(group) {
        var group_id = group.id
        var teacher_id = 5;
        var groupName = group.title;
        var json = JSON.stringify(group);

        fetch('https://localhost:44384/StudentsGroups/StudentsGroups?groupId=' + group_id
+ '&teacher_id='+ teacher_id + '&groupName=' + groupName + '&json=' + json,
            {
                method: 'PUT'
            }
        )
        .then(response => response.text())
        .catch(error => {console.error(error);})
    },
    delGroup(group) {
        var id = group.id;
        console.log(id)
    },
    drawGroups(group) {
        this.$refs.vuechart.forEach(

```

```
        function(item) {
            item.renderChrt(group)
        }
    )
}
}
</script>
<style>
.del_stud_btn {
    margin-left:2em;
}
.student_name {
    margin-left:2em;
}
.save_group {
    margin-top: 15px;
}
.remove_subgroup {
    margin-left: 2em;
}
ul {
    list-style-type: none;
}
.group_title {
    font-size: 20pt;
}
.subgroups-div {
    margin-left: 2em;
}
.subgroup_graphs {
    display: inline-block;
}
p input {
    border: none;
    display: inline;
    font-family: inherit;
    font-size: 16pt;
    font-weight: bold;
    padding: none;
```

```

    width: auto;
  }
</style>
<template>
  <div>
    <select @change="$emit('on-change', $event)">
      <option selected disabled>Выберите группу</option>
      <option v-for="group in groups" :value="group.id" v-
bind:key="group.groupId">{{group.title}}</option>
      <option value="add_new_group">+ Добавить новую группу</option>
    </select>
    <GroupItem class="container" v-for="group in groups"
      v-bind:key="group.groupId"
      v-bind:group="group"
      v-bind:all_students="all_students"
      @on-delete="$emit('on-delete', group.id)"/>
  </div>
</template>
<script>
import GroupItem from '@/components/GroupItem'
export default {
  props: ['groups', 'all_students'],
  components: {
    GroupItem
  }
}
</script>

<style>
.del_group {
  margin-right: 2em;
}
</style>

```