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

Заведующий кафедрой
к.т.н., доцент
М.С. Воробьева

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
магистра

РАЗРАБОТКА СИСТЕМЫ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗАИСХОДНОГО КОДА ПРОГРАММ

02.04.03 Математическое обеспечение и администрирование информационных систем
Магистерская программа «Разработка, администрирование и защита вычислительных систем»

Выполнил работу
студент 2 курса
очной формы обучения
Аврискин Михаил Владимирович

Научный руководитель
к.т.н., доцент
Воробьева Марина Сергеевна

Рецензент
Аналитик-программист
ООО «Геомеханические системы»
Боганюк Юлия Викторовна

Тюмень
2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для улучшения качества образования всё чаще используются различные информационные технологии, в том числе для анализа работ студентов. В то время, как программные средства анализа не могут заменить квалифицированного преподавателя, наличие сервиса для анализа сбора и кода может помочь преподавателю в поиске типичных недочетов, а также помочь учитывать большое количество работ [1].

Особенно актуально это может быть, когда проверка работ производится несколькими преподавателями в независимых группах. С применением автоматизированных систем в таком случае появляется возможность рассматривать и сравнивать работы всех студентов.

Наличие сервиса для сбора и анализа работ позволит преподавателю быстро находить предыдущие работы того же студента и сравнивать работу студента по определенному заданию с решениями других студентов по тому же заданию (попарно сравнивать файлы построчно или сравнивать большие наборы используя различные метрики — длину кода, количество методов, количество комментариев, длины идентификаторов, количество аргументов функций и др.).

Одним из возможных применений анализа кода является проверка на плагиат. По причине простоты замены имен переменных и других идентификаторов в коде программы традиционный текстовый поиск и сравнение не позволяют выявлять случаи плагиата. Поэтому, качественная проверка должна опираться на структуру кода, для чего требуется анализ кода.

В рамках выпускной квалификационной работы была поставлена цель — разработка системы интеллектуального анализа исходного кода программ.

Для достижения цели были выделены следующие задачи:

- Рассмотреть возможности существующих инструментов анализа программного кода.

- Рассмотреть научные статьи, рассматривающие вопросы анализа программного кода.
- Рассмотреть стандарты и рекомендации по написанию программного кода.
- Провести анализ методов и технологий, направленных на оценку качества программного кода и проверки кода на плагиат.
- Разработать web-сервис для анализа кода программ на примере решений лабораторных работ по курсам «Языки программирования» и «Объектно-ориентированное программирование».
- Произвести интеграцию сервиса в систему «Цифровой след студента», разрабатываемой для помощи в организации и анализе работ студентов в течение учебного процесса.

ГЛАВА 1. ОБЗОР СВЯЗАННЫХ РАБОТ И РЕШЕНИЙ

1.1. Программные пакеты анализа кода

Для анализа кода в первую очередь используется статический анализ кода — анализ исходного кода без компиляции и запуска. При этом инструменты зачастую используют правила, прописанные заранее разработчиком. Одним из менее исследованных направлений является интеллектуальный анализ кода — поиск закономерностей и правил в больших наборах данных с использованием машинного обучения. Интеллектуальный анализ кода позволяет создавать критерии оценки без значительных усилий со стороны разработчика.

Одним из первых продуктов для анализа считается Unix-утилита «lint» [2], выпущенная в 1978 году и выполняющая анализ исходного кода на языке C. В настоящее время слова lint и linter стали применяться для описания любых утилит для анализа кода на возможные критические ошибки и соблюдение стиля, а также для описания самого процесса анализа (в качестве глаголов «linting», «to lint»). Особо актуальны они оказались для интерпретируемых языков, т.к. в таких языках отсутствует шаг компиляции. Как пример можно привести инструменты JSLint и ESLint для языка JavaScript и pylint для языка Python. Эти библиотеки, а также инструменты статического анализа, встроенные в среды разработки ориентированы на использование индивидуально разработчиками.

Для автоматизации рецензирования кода, анализа программ на соответствие стандартам и поиска уязвимостей выпускаются в командных проектах используются различные коммерческие программные пакеты, интегрируемые, как правило, с системами контроля версий кода. Наиболее известными являются SonarQube (более 20 поддерживаемых ЯП), CodeSonar (C, C++, C#, Java, Python), PVS-Studio (C, C++, C#, Java), DeepCode (C, C++, Java, Javascript, typescript & Python).

Данные продукты направлены на использование в больших проектах. Так, разработчик анализатора PVS-Studio рекомендует в одной из статей задуматься об использовании статического анализа кода в проектах, трудозатраты на которые превышают 30 человеко-месяцев.

Существуют анализаторы, построенные для одного конкретного проекта. Например, scripts/checkpatch.pl — скрипт на языке perl длиной более 6000 строк, используемый для проверки нового кода для ядра Linux на соблюдение набору правил.

При всем разнообразии программных продуктов для статического анализа ориентируемым на академическую сферу можно считать только пакет Gauntlet. Пакет использовался на курсах введения в программирование на языке программирования Java в военной академии США. Его задачей было нахождение типичных синтаксических ошибок и создание отчетов, более понятных начинающим программистам по сравнению со стандартными средствами компиляции [3].

Некоторые инструменты анализа не анализируют исходный текст программы, а его компилированный вариант, например, LLVM-код. В таком случае инструмент может обладать более широкой поддержкой языков программирования, т.к. будет возможен анализ программ на любом из языков, компилируемых в LLVM. Недостатком таких инструментов является отсутствие возможности проверки исходного кода. Таким образом, возможен анализ возможного неопределенного поведения программы, но невозможен анализ соблюдения формата исходного кода.

Кроме статического анализа кода в разработке также применяется динамический анализ кода — анализ работы программы при запуске. Наиболее распространенным примером анализа является создание юнит-тестов. Преимуществом динамический анализа является проверка на реальных данных, недостатком — невозможность покрытия всех возможных путей кода. Также динамический анализ кода используют при автоматической

проверке учебных задач в МООК системы Stepik и на соревнованиях по спортивному программированию. В этой ситуации используется набор тестов для проверки, подготовленный для одной конкретной задачи и применяющийся только для неё. При этом исчезает привязка к языку программирования — анализировать можно любой исполняемый код, соблюдающий требования по формату ввода и вывода.

На платформах, посвященных спортивному программированию, динамический анализ является первоочередным и применяется ко всем решениям задач. Количество решений при этом за несколько часов соревнования может превосходить сотни тысяч. В рамках теста для входных данных у решения участника проверяются отсутствие ошибок выполнения, верность выходных данных и отсутствие превышения ограничения лимитов потребления памяти и времени.

В дополнение к динамическому анализу кода на платформе CodeForces к коду на языке C++ также применяется статический анализ кода с использованием библиотеки Clang, выявляющий потенциальные ошибки и указывающий на причины их возникновения (см. рисунок 1). На платформе после соревнований также происходит поиск похожих решений для проверки на наличие плагиата, и имеется возможность наглядного сравнения разницы двух версий кода.

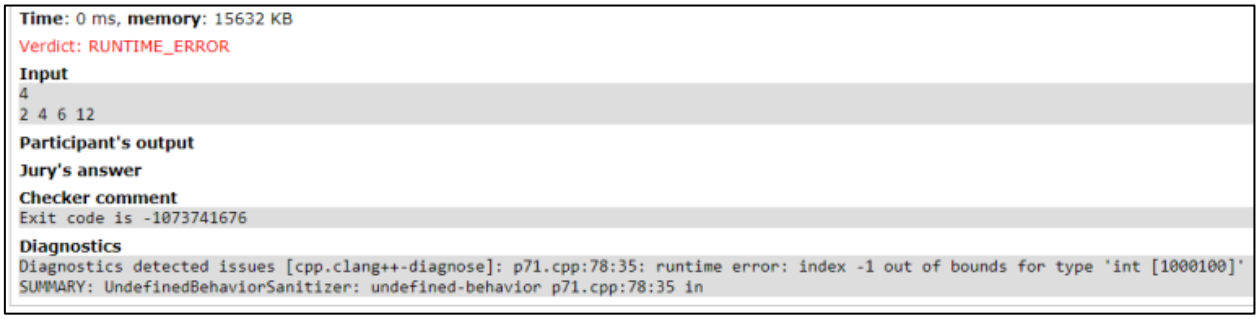

Рисунок 1. Пример результата диагностики на платформе Codeforces

Инструменты для проверки кода на плагиат также используются при проверке кода при собеседованиях на платформе HackerRank (см. рисунок 2).

Для проверки используется сервис MOSS (Measure Of Software Similarity, мера схожести исходного кода), предоставляемый как онлайн-сервис Стэнфордским университетом [4].

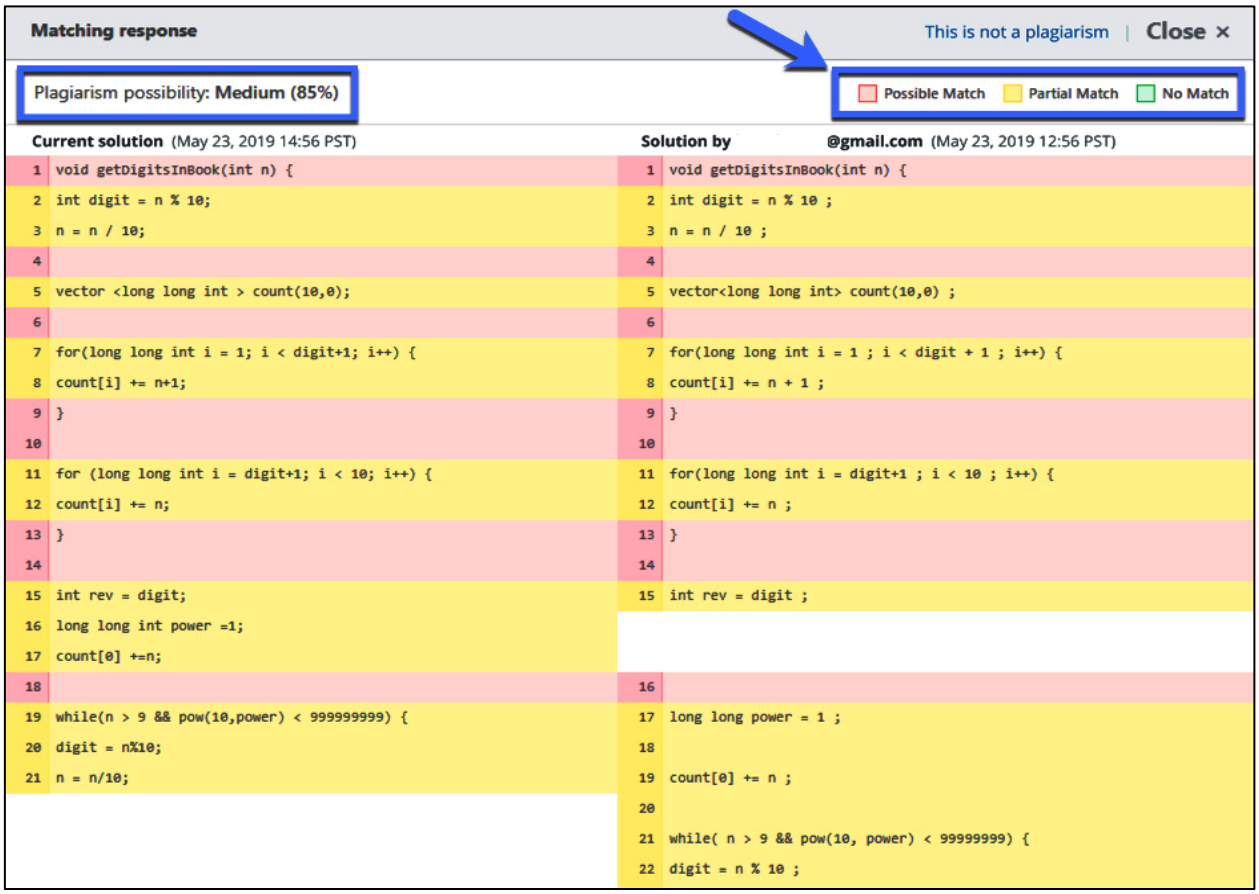

Рисунок 2. Сравнение двух образцов исходного кода на платформе HackerRank

Для тестирования работоспособности пакетов поиска ошибок в исходном коде зачастую используются известные программные продукты с открытым исходным кодом, например, Firefox и Return to Castle Wolfenstein [5], или случайные наборы проектов с платформ хостинга IT-проектов GitHub, GitLab и Bitbucket.

1.2. Подходы к решению задачи анализа программного кода в научных работах

В большинстве научных статей по интеллектуальному анализу кода рассматривается возможность использования стилометрии кода для определения авторства кода и проверке на плагиат.

Примерами таких работ являются:

- А.В. Куртукова, А.С. Романов, «Идентификация автора исходного кода методами машинного обучения» (2019, ТУСУР) [6],
- Ningfei Wang, Shouling Ji, Ting Wang «Integration of Static and Dynamic Code Stylometry Analysis for Programmer De-anonymization» (2018, Лихайский университет) [7],
- Стрельчёнок Г.В. «Ступенчатый метод проверки исходного кода программы на плагиат» (2016, Санкт-Петербургский государственный университет) [8],
- Aylin Caliskan-Islam et al. «De-anonymizing programmers via code stylometry» (2015, Дрексельский университет и др.) [9],
- IvanKrsul, Eugene H. Spafford «Authorship Analysis: Identifying The Author of a Program» (1995, Университет Пердью) [10].

Интеллектуальный анализ программного кода играет важную роль в помощи написания и улучшения программного кода, борьбе с плагиатом, аудите программного кода. Автоматизированный анализ может использоваться преподавателем при проверке студенческих работ, что может послужить улучшению качества процесса обучения.

Методы, приведенные в статье «Ступенчатый метод проверки исходного кода программы на плагиат» [8], подразумевают попарное сравнение образцов исходного кода и не подходят для анализа больших объемов данных в связи с производительностью подхода. Также, в отличие от алгоритмов определения авторства с использованием стилометрии имеется возможность сравнения лишь с решениями автора конкретных задач, а не

стилем автора в целом. Другими словами, два образца кода, написанные одним автором для разных задач будут считаться абсолютно разными.

Однако, эти методы могут быть использованы для более наглядного сравнения похожих образцов кода после устранения заведомо различных образцов другими методами.

Методы, описанные в других статьях, опираются на использование стилометрии кода и дают возможность определять авторство исходного кода даже если автор решал абсолютно другую задачу и при попарном сравнении образцы исходного кода являются различными.

В целом структура подхода к задаче определения с использованием стилометрии является неизменной. По образцам исходного кода выделяются стилометрические признаки, описанные в разделе 2.1, а затем производится классификация типичными методами машинного обучения [11].

Статьи отличаются выбором различных языков и инструментов их анализа, а также наборами данных, на которых проводились исследования.

В статье «Authorship Analysis: Identifying The Author of a Program» [10] производится анализ программного кода сотрудников и студентов университета Пердью, написанного на языке C. В качестве признаков выбран набор признаков форматирования и частоты использования ключевых слов. После этого из набора признаков были выбраны более существенные и была произведена классификация методом линейного дискриминантного анализа. Стоит отметить, что во время написания статьи среды разработки в меньшей степени сопутствовали форматированию кода, что давало большую возможность использовать признаки форматирования в целях определения авторства.

В статье «De-anonymizing programmers via code stylometry» [9] используется анализ нечетких абстрактных синтаксических деревьев (АСД) для выделения признаков кода на языках C и C++ с дальнейшей классификацией с использованием случайного леса. Тестирование

производилось на наборах программного кода с соревнований по программированию Google Code Jam. Дополнительно произведена попытка анализа программного кода на Python.

В статье «Integration of Static and Dynamic Code Stylometry Analysis for Programmer De-anonymization» [7] производится анализ программного кода на языке Python и в дополнение к признакам, выделяемым статическим анализом кода, используются признаки, выделяемые динамическим анализом — время работы программы и статистика обращения к памяти. Классификация производится с использованием глубоких нейросетей, а тестирование — на наборах кода Google Code Jam.

В работе «Идентификация автора исходного кода методами машинного обучения» [6] производится анализ множества проектов, взятых с сервиса GitHub, анализируется код на 8 языках программирования и классификация с использованием гибридной сверточно-рекуррентной нейронной сети.

Проблемой во многих из работ по определению авторства может являться выбор данных для тестирования. В случае использования исходного кода реальных проектов с открытым исходным кодом возникает проблема формального определения понятия авторства в работах нескольких авторов.

В случае использования программ с онлайн-платформ соревнований по спортивному программированию (Google Code Jam, Codeforces и др.) в образцах кода авторами могут быть использованы заготовки кода и шаблоны.

Заготовка кода, фрагмент которой показан на рисунке 3, занимает более 90% объема кода и присутствует почти во всех образцах кода автора. А у некоторых авторов в явном виде указывается имя или другая информация об авторстве. Аналогично, некоторые проекты ПО с открытым исходным кодом включают текст лицензии в начале каждого из файлов.

Очевидно, в таком случае программа определения авторства фактически будет сравнивать параметры этих повторяющихся шаблонов, а не стиль написанного для отдельной задачи кода.

```
_MFadjlist[right].insert(make_pair(left,make_pair(e,false))); } else e =
_MFedgehash[make_pair(left,right)]; _MFflow[e] = 0; _MFcapacity[e] = cap;
_MFcost[e]=cost;}

private:
  set<pair<int,pair<int,bool> > > _MFadjlist[MF_VERTEX_SIZE];
  map<pair<int,int>,int> _MFedgehash;
  T _MFflow[MF_EDGE_SIZE], _MFcapacity[MF_EDGE_SIZE];
  S _MFcost[MF_EDGE_SIZE];
  int _MFparent[MF_VERTEX_SIZE], _MFsnkdist[MF_VERTEX_SIZE];
  pair<int,bool> _MFpedge[MF_VERTEX_SIZE];
};
// END CUT - Max Flow

//BEGIN CUT - Set FUNCTIONS O(N)
template<class T>
void setIntersect(const set<T> &A, const set<T> &B, set<T> &C){
set_intersection(A.begin(),A.end(),B.begin(),B.end(),inserter(C,C.begin()));
}
template<class T>
void setDifference(const set<T> &A, const set<T> &B, set<T> &C){
set_difference(A.begin(),A.end(),B.begin(),B.end(),inserter(C,C.begin()));}
//END CUT - Set FUNCTIONS
```

Рисунок 3. Фрагмент заготовки кода, использованного в соревнованиях по программированию

Анализ стилистических признаков кода, рассмотренных в работах, может быть применен не только для определения авторства, но и для визуализации изменения стиля кода студента в течение времени и оценке объективных и субъективных метрик программного кода.

ГЛАВА 2. МЕТРИКИ ПРОГРАММНОГО КОДА

2.1. Стилометрические признаки для решения задачи классификации

При решении задач статистического и кластерного анализа работа с необработанным программным кодом может требовать чрезмерно больших затрат ресурсов. В целях снижения ресурсоёмкости используется конструирование признаков исходного кода.

Наиболее распространенным подходом является использование стилометрии кода. По полученным векторам стилометрических признаков оказывается возможной быстрая классификация и кластеризация исходных текстов используя типичные инструменты машинного обучения.

Признаки кода можно разделить на такие категории как: лексические признаки, признаки форматирования, синтаксические признаки, стилевые признаки и качественные признаки.

К лексическим признакам можно отнести факт использования тех или иных лексем, их длины, и частоты:

- количество литералов;
- наиболее часто встречающиеся лексемы и их частоты;
- количество ключевых слов (if, while, def и др.);
- средняя длина лексем;
- количество комментариев;
- количество функций;
- среднее количество аргументов функций;
- среднее отклонение количества аргументов функций;
- средняя длина строки;
- среднее отклонение длин строки.

К признакам форматирования можно отнести признаки расстановки непечатных знаков:

- наличие переноса строки перед фигурными скобками;
- количество символов пробела;
- количество символов табуляции;
- количество пустых строк;
- количество непечатных символов;
- использование знаков табуляции вместо пробелов в начале строки.

Для получения синтаксических признаков необходимо, чтобы код был синтаксически корректным, т.е. код должен следовать синтаксическим правилам языка программирования. В таком случае строится абстрактное синтаксическое дерево (АСД) — дерево, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья — с соответствующими операндами, и параметры данного дерева используются как количественные признаки.

Для примера конструирования синтаксических признаков рассмотрим дерево, полученное из кода на языке C++ с помощью библиотеки Joern (см. рисунок 4).

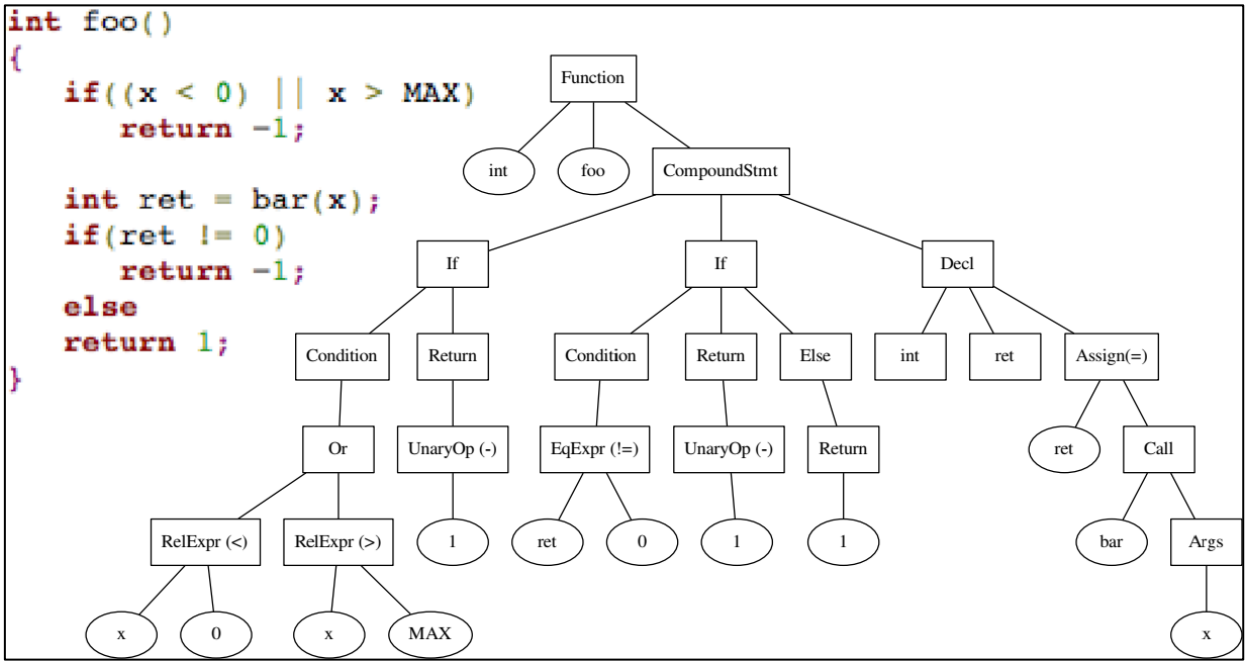

Рисунок 4. Пример абстрактного синтаксического дерева

Из построенного дерева выделяются следующие синтаксические признаки:

- частоты юниграмм и их средние глубины.
- высота АСД;
- средняя глубина для каждого из 58 типов узлов, не являющихся листами;
- частоты 84 ключевых слов C++;
- частоты узлов, не являющихся листами (TF и TF-IDF меры);

При конструировании признаков кода, написанного на других языках программирования, процесс выполняется аналогично, количество синтаксических признаков зависит от лексики языка.

Кроме синтаксических признаков, можно выделить стилевые и качественные признаки программного кода [12].

К стилевым признакам отнесем признаки, рекомендуемые, но не обязательные при написании кода, такие как:

- количество лексем, состоящих из английских слов;
- количество функций, названых глагольными словами;
- количество классов, названых существительными словами;
- использование snake case и camel case в лексемах, состоящих из нескольких слов.

К качественным признакам можно отнести степень несоблюдения правил написания удобочитаемого кода:

- длина лексем;
- длина тела функций и описаний классов;
- использование неименованных констант;
- написание функций с большим количеством аргументов.

Также можно рассмотреть числовые оценки программного обеспечения:

- цикломатическая сложность;
- метрики программного пакета от Роберта Сесиль Мартина;
- связность;
- индекс удобства сопровождения;
- ABC-метрика;
- метрики Хальстеда:
    - словарь программы;
    - аппроксимированная длина программы;
    - объем;
    - сложность;
    - трудозатраты.

Таким образом, выделяются более 30 признаков, 6 из которых являются векторами количественных признаков. Если привести их к набору числовых значений, получится несколько сотен признаков, обработка которых будет слишком ресурсоемкой.

Поэтому, при решении задачи классификации для дальнейшей обработки проводится отбор этих признаков по критерию энтропии, используя обучающий набор входных текстов. При решении задачи качественной оценки кода необходим выбор ограниченного набора признаков вручную.

2.2. Использование признаков кода для качественной оценки

Международный стандарт ISO/IEC 9126 «Software engineering — Product quality» разделяет параметры качества кода на 6 категорий, каждая из которых разделена на подкатегории:

- Функциональные возможности (Functionality).
    - Соовместимость (Suitability).

    - Точность (Accuracy).
    - Заменяемость (Interoperability).
    - Безопасность (Security).
    - Функциональное соответствие (Functionality compliance).
- Надежность (Reliability).
    - Зрелость (Maturity).
    - Отказоустойчивость (Fault tolerance).
    - Способность к восстановлению (Recoverability)
    - Соответствие стандартам надежности (Reliability compliance).
- Практичность (Usability).
    - Понятность (Understandability).
    - Удобство обучения (Learnability).
    - Удобство работы (Operability).
    - Привлекательность (Attractiveness).
    - Соответствие простоте использования (Usability compliance).
- Эффективность (Efficiencesy).
    - Временная эффективность (Time behaviour).
    - Ресурсоемкость (Resource utilization).
    - Соответствие стандартам производительности (Efficiency compliance).
- Сопровождаемость (Maintainability).
    - Анализируемость (Analyzability).
    - Простота изменения (Changeability).
    - Стабильность (Stability).
    - Простота проверки (Testability).
    - Соответствие сопровождаемости (Maintainability compliance).
- Мобильность (Portability).
    - Адаптируемость (Adaptability).
    - Удобство установки (Installability).

- Способность к сосуществованию (Co-existence).
- Заменяемость (Replaceability).
- Соответствие стандартам переносимости (Portability compliance).

Российским аналогом стандарта является ГОСТ 28195-89 «Оценка качества программных средств» [13]. Оба стандарта описывают методы подхода к оцениванию широкого диапазона ПО, однако детали оценки более конкретных продуктов остаются за командой разработчиков.

Более строгие методы оценки присутствуют в рекомендациях по разработке ПО для критических систем и систем реального времени.

Например, стандарт MISRA C:2004 (Motor Industry Software Research Association) включает такие правила, как правила, указанные в источнике [14]:

2.2. *Допускается использование комментариев «/* … */», комментарии вида «//» не допускаются.*

2.4. *Фрагменты программного кода не должны быть закомментированы.*

5.2. *Переменные во внутреннем блоке видимости не должны иметь то же имя, что и переменные во внешнем блоке видимости.*

6.1–6.2 *Переменные типа char должны использоваться только для символьных значений, типа signed char и unsigned char — для числовых.*

А набор правил «The power of 10», разработаный NASA, содержит такие правила, как правила, указанные в источнике [15]:

2. *У всех циклов должна быть явно задана верхняя граница*

4. *Каждая из функций должна умещаться на листе бумаги в стандартном формате по одной строке на объявление или выражение. Как правило — это означает использование не более 60 строк кода на функцию.*

Некоторые из этих правил сложно применимы для кода общего назначения, но некоторые из них вполне можно брать за метрики качества кода.

Добавляя к подобным стандартам руководства, предназначенные для отдельных языков программирования, такие как PEP 8 [16] и руководство по программированию на C# от Microsoft можно получить набор признаков кода, отражающих качество.

ГЛАВА 3. ИНСТРУМЕНТАЛЬНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА, ИСПОЛЬЗОВАНЫЕ В РАЗРАБОТКЕ СИСТЕМЫ

Для работы с программным кодом было спроектировано и разработано веб-приложение, позволяющее наглядно просматривать работы студента и сравнивать их с другими студентами.

Большая часть разработки проводилась с использованием языков программирования C#, HTML, JavaScript в интегрированной среде разработки Microsoft Visual Studio.

Для разработки основного веб-приложения использован Фреймворк «ASP.NET Core» с использованием технологии «Razor Pages» для генерации HTML-страниц.

Для хранения данных использована СУБД PostgreSQL и библиотека Npgsql для связи веб-приложения с СУБД.

Исходные коды студентов для обработки взяты с портала elearning.utmn.ru — системы электронного обучения ТюмГУ, основанной на платформе Moodle. Для графической визуализации использована JavaScript-библиотека Highcharts. Для отображения программного кода на веб-страницах с подсветкой синтаксиса используется библиотека «highlight.js».

Современные версии Visual Studio помимо поддержки веб-разработки в рамках собственных фреймворков включает модули для поддержки разработки с использованием сторонних популярных продуктов таких как Apache Cordova, Angular, jQuery, Bootstrap, Django, Backbone.js и Express. Поддерживается интеграция с менеджером пакетов npm и веб-сервером node.js. Поддерживается подсветка и проверка корректности синтаксиса HTML, JavaScript и CSS. Таким образом Visual Studio обладает полным набором функционала, необходимого для веб-разработки.

Фреймворк «ASP.NET Core» является развитием более старого «Active Server Pages для .NET», построенном на кроссплатформенной платформе с

открытым исходным кодом «.NET Core». Фреймворк позволяет создавать веб-приложения на языке C# с возможностью запуска на серверах с ОС Windows, Linux и macOS. По сравнению с устаревшим «ASP.NET» он обладает большей производительностью и расширяемостью. Фреймворк поддерживает большинство модулей, реализованных для платформы «.NET Core» таких как модули работы с базами данных, модули для машинного обучения, модули для работы с различными файловыми форматами.

Несмотря на превосходство платформы .NET Core над .NET Framework, последняя будет получать поддержку в течение долгого времени, т.к. она является частью ОС Windows [17].

«Razor Pages» — технология, рекомендуемая для генерации HTML-страниц в приложениях на ASP.NET. Технология позволяет создавать шаблоны страниц в CSHTML-файлах, совмещающих синтаксис HTML и C#.

Для создания страниц при этом, как правило, используется связка из C#-файла с расширением .cs и CSHTML-файла с тем же именем и расширением .cshtml. В файле .cs императивно описываются действия, производимые при HTTP запросах на сервер, а в CSHTML-файле содержится шаблон для верстки веб-страницы с возможностью обращения к данным, посчитанным в методах, описанных в .cs файле.

PostgreSQL — кроссплатформенная СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций такие как сложные запросы, внешние ключи, триггеры, изменяемые представления, транзакционную целостность и многоверсионность [18].

PostgreSQL был выбран по причине производительности, сопоставимой с MySQL, более гибкой работой с неструктурированными данными и наличием оконных функций.

Moodle — система управления курсами, являющаяся веб-приложением и позволяющая преподавателям создавать учебные курсы и задания, а

студентам — выполнять данные задания. Система написана на языке PHP. Для работы используется PHP сервер и реляционная база данных, как правило, MySQL или PostgreSQL.

Для разработки системы анализа программного кода были использованы работы сдаваемые в систему Moodle студентами первых курсов направлений «Компьютерная безопасность» и «Математическое обеспечение и администрирование информационных систем» Института математики и компьютерных наук ТюмГУ, Система позволяет выгружать все данные по курсам в формате MBZ, являющимся GZIP-архивом с файлами курса и XML-метаданными файлов и курса. Из данного архива веб-приложение извлекает файлы с исходным кодом и их метаданные. Было использовано около 600 работ, написанных на языках C# и C++.

Highcharts — JavaScript-библиотека для отображения графиков и диаграмм во множестве различных форматах: гистограммы, круговые диаграммы, линейные графики, лепестковые диаграммы, диаграммы размаха, таблицы, графы. Библиотека позволяет добавлять интерактивные, анимированные графики на сайт или в веб-приложение. Для использования импортируются JavaScript-файлы, нужные для выбранного графика, создается DOM-контейнер и вызывается функция отрисовки с параметрами графика. Параметры включают в себя тип диаграммы, описание координатных осей, описание подписей и задание данных. Все параметры передаются в виде JavaScript-объекта (см. рисунок 5).

```
{
    chart: { type: 'scatter', zoomType: 'xy'},
    title: { text: 'Зависимость стилистических признаков от балла ЕГЭ'},
    height: 600,
    xAxis: {
        title: {
            enabled: true,
            text: 'Средняя длина строки',
            style: { fontSize: "19pt", color: "#000"}
        },
        startOnTick: true,
    },
    series: [{ name: 'Набор 1', data: scatter_zip[0] }
             { name: 'Набор 2', data: scatter_zip[1] },]
}
```

Рисунок 5. Фрагмент параметров для отрисовки графика в highcharts

Для задач машинного обучения использовалась библиотека ML.NET — фреймворк для научных вычислений и машинного обучения с открытым исходным кодом на платформе .NET. Фреймворк включает в себя набор библиотек для обработки данных, визуализации и машинного обучения.

ГЛАВА 4. ПРОГРАМНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ АНАЛИЗА ПРОГРАММНОГО КОДА

4.1. Источники данных и их обработка

Для загрузки файлов из курсов в системе Moodle был рассмотрен формат хранения файлов при выгрузке данных курсов из системы. MoodleFile представляет из себя tar GZIP архив, содержащий набор загруженных пользователями файлов с именами, соответствующими шестнадцатеричному представлению SHA-1 хеш-суммы содержимого файлов, а также информацию об этих файлах и о курсе, структурированную в виде XML-файлов. К файлам, загруженным пользователями, относятся исходные коды программ, документы с отчетами о проделанной работе и загруженные преподавателем файлы заданий. Пример метаданных по файлу показан на рисунке 3.

Для анализа файлов необходимо извлечь их из исходного архива, считать список файлов и их метаданные из файла «files.xml», считать данные самих файлов. В дополнение также нужно считать данные о заданиях (такие как названия лабораторных работ и их описание) и учебных группах, сохраненные в файлах «assign.xml» и «groups.xml» соответственно.

Для работы с данными Moodle были разработанные классы (см. Табл.1).

Таблица 1. Описание классов

Имя класса	*Назначение*
SHAUtils	Вспомогательный класс, считающий SHA-1 сумму данных и возвращающий её в виде массива байт и шестнадцатеричной строки.
MoodleFile	Хранение метаданных о файле из системы Moodle

Имя класса	*Назначение*
MoodleActivity	Хранение данных о задании из системы Moodle
MbzArchive	Извлечение информации из архивов резервных копий Moodle

Для хранения метаданных по файлу работы и его содержимого был реализован вспомогательный класс MoodleFile (см. рисунок 6), а для хранения данным по заданиям — класс MoodleActivity.

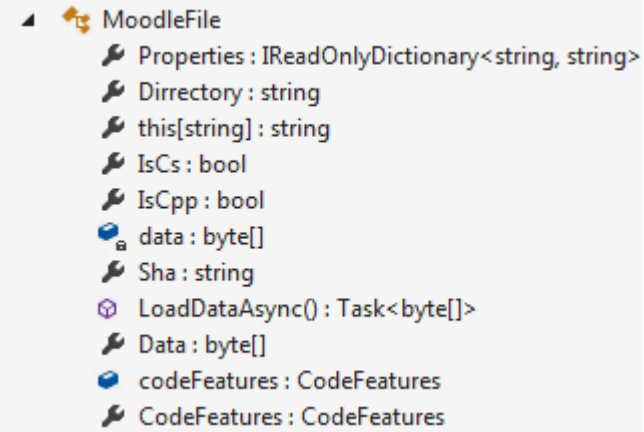

Рисунок 6. Список членов класса MoodleFile

Класс MoodleFile содержит свойства файла в виде словаря, байт-массив, соответствующий данным, а также вспомогательные свойства и метолы для более удобного доступа к данным.

Для чтения всего архива реализован класс MbzArchive с конструктором, принимающим на вход путь к файлу архива и сохраняющим данные из архива в виде списка экземпляров класса MoodleFile, и списком экземпляров класса MoodleActivity.

Чтение архива осуществляется с помощью библиотеки SharpCompress, чтение XML-файлов — с помощью класса System.Xml.XmlReader, входящего в состав платформы .NET.

```
<file id="190784">
<contenthash>12a54b075202968d7ea583b046c8815b4d030f84</contenthash>
    <contextid>63433</contextid>
    <component>assignsubmission_file</component>
    <filearea>submission_files</filearea>
    <itemid>40775</itemid>
    <filepath>/</filepath>
    <filename>Иванов_Иван_Лаб1.cs</filename>
    <userid>12214</userid>
    <filesize>12520</filesize>
    <mimetype>application/x-csh</mimetype>
    <timecreated>1583162043</timecreated>
    <source>Иванов_Иван_Лаб1.cs</source>
    <author>Иванов Иван</author>
    <reference>$@NULL@$</reference>
  </file>
```

Рисунок 7. Метаданные пользовательского файла, выгруженного из системы Moodle

Помимо загрузки файлов из Moodle возможна отправка файлов с исходным кодом через web-интерфейс. Для этого пользователь выбирает автора работы и задание из списка и загружает файл с исходным кодом.

Для автоматизированных загрузок также возможна отправка архива с файлами, названными по заданному шаблону.

Разработанный шаблон имеет вид:

$$userId\_AssigmentId\_FileName.ext \qquad (1)$$

где

userId — идентификатор студента в базе данных,

AssigmentId — идентификатор задания в базе данных,

FileName.ext — исходное имя файла.

По названиям файлов определятся студент и задание для каждого из файлов, а при отсутствии идентификаторов в базе данных отобразится отчет по каждому из файлов, которые не удалось загрузить.

Пример страницы загрузки архива показан на рисунке 8. В полном отчете отображается информация по каждому из файлов — имя студента, название задания и статус загрузки. В случае отсутствия идентификатора студента или задания в базе данных отображается соответствующая ошибка.

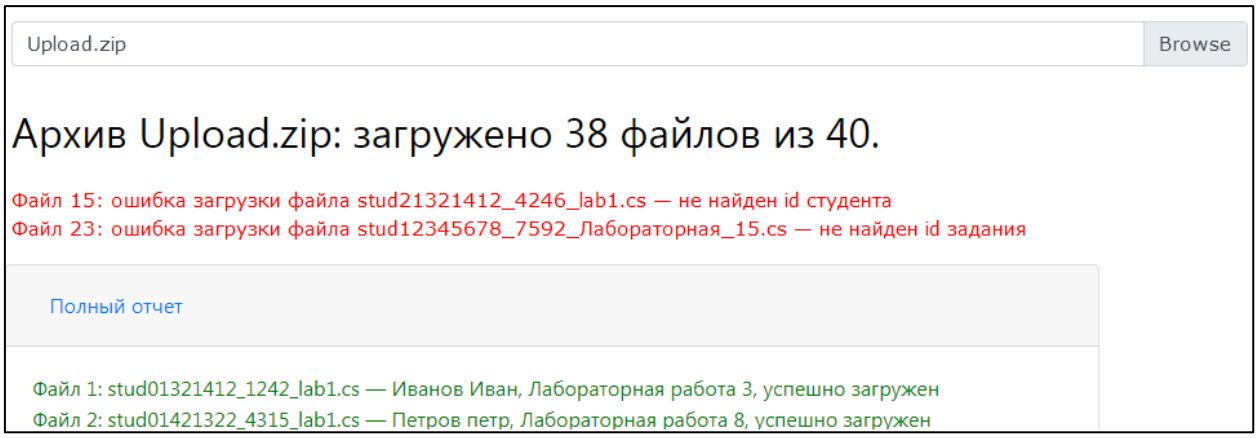

Рисунок 8. Интерфейс пакетной загрузки работ

4.2. Хранение данных

После загрузки образца программного кода текст и метаданные сохраняются в базу данных в исходном виде. Сохранение исходных данных позволяет пересчитывать при дальнейшей необходимости извлекаемые признаки, а также позволяет преподавателю просматривать исходный код.

Для выделения признаков кода был создан класс CodeFeatures (см. рисунок 9), экземпляр которого создается из файла с исходным кодом.

Для обработки различных языков созданы производные классы, такие как CsCodeFeatures и CppCodeFeatures, считающие разные вектора признаков.

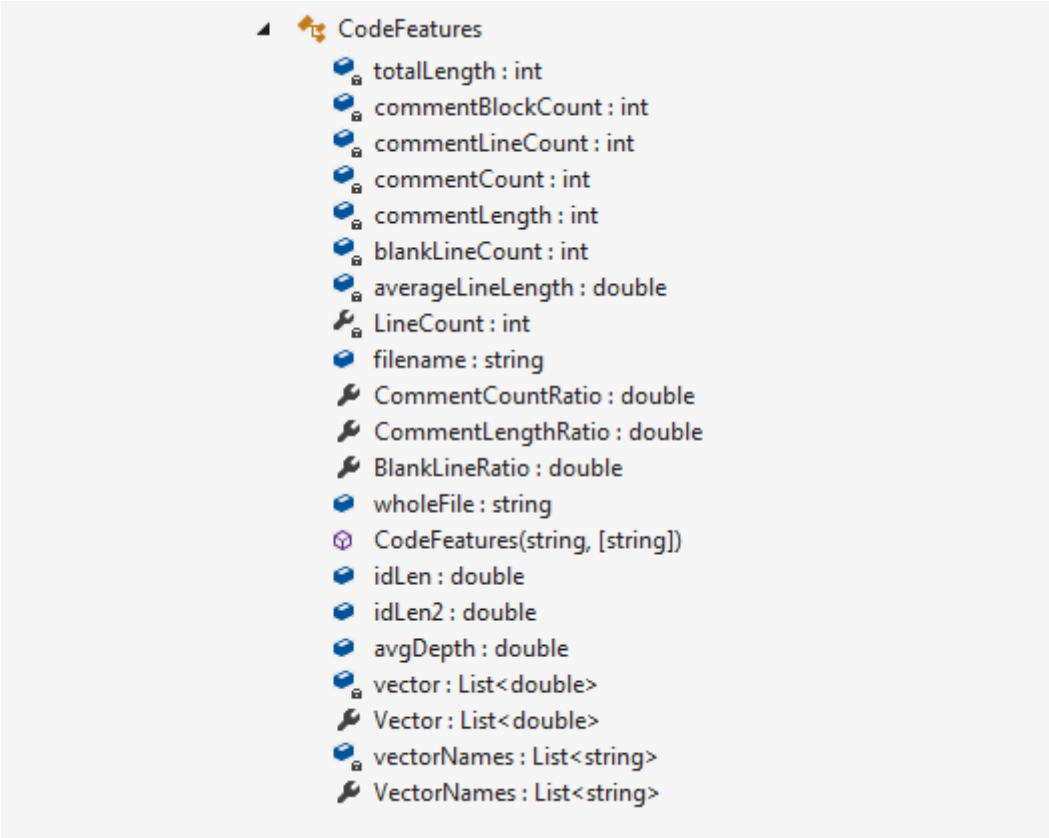

Рисунок 9. Список членов класса CodeFeatures

При создании экземпляра класса с помощью библиотеки Microsoft.CodeAnalysis.CSharp строится абстрактное синтаксическое дерево, которое используется для нахождения численных значений искомых признаков:

- Количество однострочных комментариев.
- Количество многострочных комментариев.
- Количество полностью закомментированных строк.
- Количество закомментированных символов.
- Количество пустых строк.
- Среднюю длину строки в символах.
- Среднюю длину метода в строках.
- Средняя вложенность фигурных скобок.
- Количество лямбда-выражений
- Общую длину кода.

Полученный вектор признаков сохраняется в базу данных для возможности быстрого анализа в дальнейшем.

Для хранения данных используется реляционная СУБД PostgreSQL. В базе данных хранится таблица с файлами в виде массива байт, таблица с метаданными файлов, таблица с вычисленными признаками, таблица с информацией о студентах, таблица с информацией о заданиях, таблица с названиями признаков, вычисляемых для различных языков программирования (см. Табл. 1–6).

Таблица 1. Описание полей таблицы students

Имя поля	*Тип поля*	*Назначение*
id	text	Идентификатор, как правило, в формате «stud12345678»
full_name	text	Полное имя студента
group	text	Номер группы студента
enrollment_year	int	Год поступления в университет
created_at	timestamp	Дата создания записи

Таблица 2. Описание полей таблицы code_metadata

Имя поля	*Тип поля*	*Назначение*
id	text	Идентификатор файла. В его качестве используется SHA-1 сумма.
student_id	text	Идентификатор студента — автора работы.
assignment_id	int	Идентификатор задания.
filename	text	Имя файла.
file_size	int	Размер файла в байтах.
language	text	Язык программирования.
created_at	timestamp	Время создания записи.
submmited_at	timestamp	Время отправки решения.

Таблица 3. Описание полей таблицы code_data

Имя поля	*Тип поля*	*Назначение*
id	text	Идентификатор файла. В его качестве используется SHA-1 сумма.
data	bytea	Содержимое файла, сжатое с библиотекой Gzip.

Таблица 4. Описание полей таблицы assignments

Имя поля	*Тип поля*	*Назначение*
id	int	Идентификатор задания.
name	text	Название задания.
description	text	Описание задания.
date	timestamp	Дата начала или окончания задания, для сортировки.

Таблица 5. Описание полей таблицы language_features

Имя поля	*Тип поля*	*Назначение*
language	text	Название ЯП
feature_number	int	Номер признака
feature_name	text	Имя признака

Таблица 6. Описание полей таблицы code_features

Имя поля	*Тип поля*	*Назначение*
id	text	Идентификатор файла. В его качестве используется SHA-1 сумма.
f1	real	Значение 1-го признака.
…	real	…
f12	real	Значение 12-го признака.

Для хранения признаков кода используется таблица code_features, содержащая 12 полей для 12 признаков типа real. В зависимости от языка программирования в таблице могут храниться разные признаки. Описание признаков для каждого из поддерживаемых языков содержится в таблице language_features.

Схема базы данных отображена на рисунке 10.

Для взаимодействия с СУБД используется библиотека Npgsql — ADO.NET провайдер для PostgreSQL с открытым исходным кодом. Библиотека также реализует провайдерами для EF Core и обладает такими функциями как асинхронным режимом работы, создания пула подключений, пакетным копированием данных.

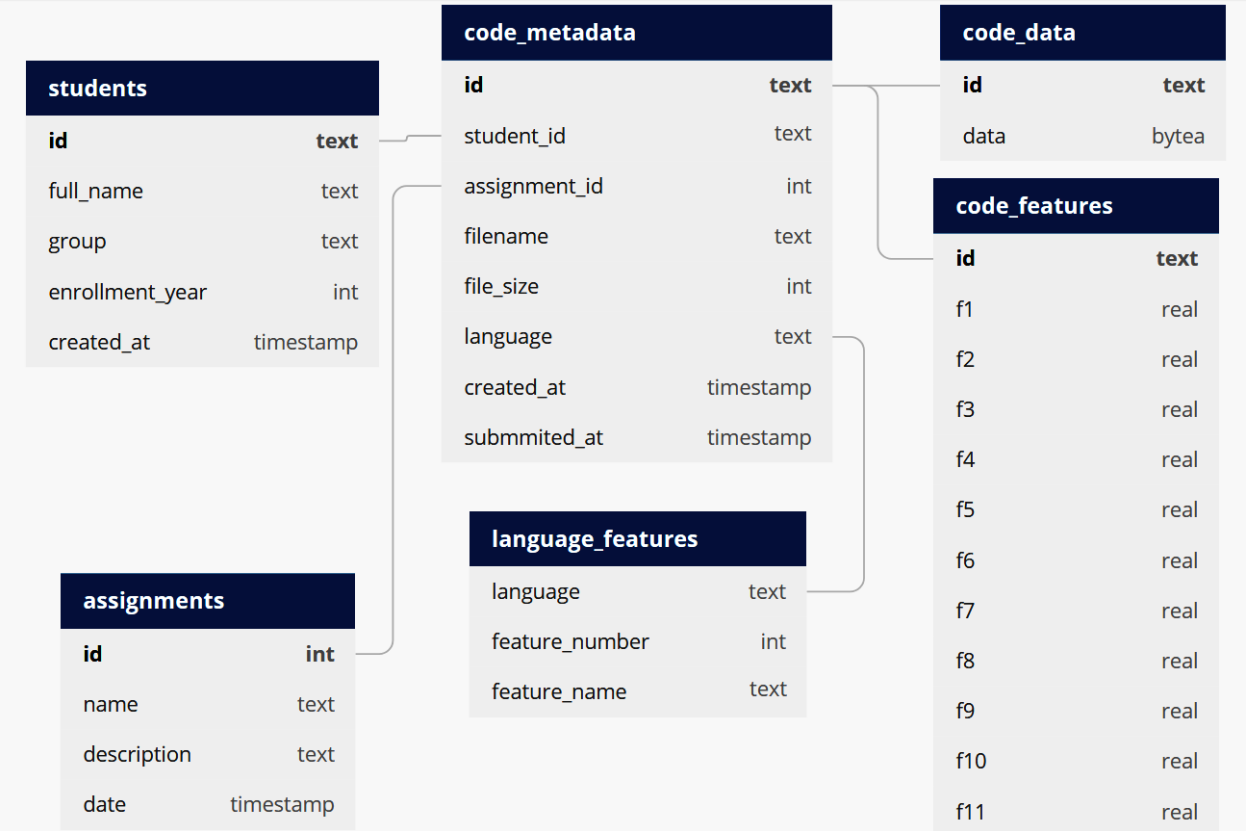

Рисунок 10. Схема базы данных для хранения работ

4.3. Веб-интерфейс

Для реализации веб-интерфейса пользователя были созданы страницы Razor pages и соответствующие классы:

- Index — стартовая страница.
- Student — страница для просмотра информации по работам студента.
- Groups — страница для просмотра статистики по группам студентов и назначения студентов по группам.
- ScatterView — страница для просмотра графиков разброса и корреляционного анализа признаков кода.
- UploadCode — страница для отправки одного образца исходного кода.
- BatchUploadCode — страница для пакетной отправки исходного кода.
- AddStudent — страница для добавления студентов в базу данных.
- AddAssignment — страница для добавления заданий в базу данных.

ГЛАВА 5. РАБОТА С ВЕБ-ПРИЛОЖЕНИЕМ

Вся работа с приложением происходит через веб-интерфейс, студенты могут отправлять свои работы по соответствующим заданиям для проведения анализа и просмотра отчета, а преподаватель может просматривать статистику по выбранному студенту или по всей группе.

5.1. Просмотр данных отдельного студента

В приложении можно просматривать код отдельной работы студента с подсветкой синтаксиса, отображением стилометрических признаков кода, а также сравнения признаков с признаками других студентов и других работ этого же студента.

Для примера визуализации взяты следующие признаки:

- Доля строк с комментариями и доля комментариев в коде — позволяют понять, что в работе может быть очень мало или много. Малое количество комментариев говорит о недостаточной самодокументации, а большое может означать что автор кода пишет нечитабельные функции, в которых сложно разобраться без подробных комментариев.
- Доля пустых строк — позволяет заметить, если программист не ставит отступы. При программировании рекомендуется разделять логические блоки в программе с помощью пустых строк.
- Средняя длина строки — позволяет заметить, в каких из экземпляров кода могут быть слишком длинные строки. Рекомендации по оформлению кода указывают на необходимость разбиение слишком длинных строк на несколько для удобочитаемости как на бумаге, так и на экране компьютера.

- Средняя длина идентификаторов — малые значения могут указывать на чрезмерное использование однобуквенных названий переменных, в то время как рекомендуется использовать в качестве названий слова, описывающие значение переменной.
- Средняя глубина вложенности фигурных скобок — большие значения указывают на слишком большую вложенность циклов, условных операторов и других конструкций. Это приводит к большим длинам строк и низкой читабельности кода в целом. Подобные блоки кода рекомендуется выносить в отдельные функции.

При открытии страницы, соответствующей образцу кода, отображается таблица со значениями признаков при выполнении других лабораторных работ данным студентом. При выборе одной из лабораторной работ, выполненных студентом, отображаются значения признаков по этой работе по сравнению с медианой и квартилями других работ по этому заданию (см. рисунок 11). На примере по таблице видно, что студент Иванов Иван, начиная с третьей работы, перестал использовать комментарии при выполнении задания, что может затруднить проверку преподавателем.

Рисунок 11. Отображение таблицы со значениями признаков работ студента

При выборе отдельного признака отображается график динамики изменения этого признака со временем, отображающий значения признака из различных лабораторных работ.

Очевидно, абсолютные значения могут отражать характер выполняемого задания, а не только развитие студента. Для того чтобы снизить степень влияния характера задания на результат, рекомендуется оценка значений признака по сравнению с значениями других студентов по этой же лабораторной работе.

Поэтому, в дополнение к графику абсолютных значений, в приложении вычисляется и визуализируется отклонение значения от среднего по учебной группе.

Также отображается гистограмма распределения признака, взятого среди всех студентов группы, и значение признака текущего студента. Это позволяет оценить стиль кода группы в целом и то, насколько стиль студента отклоняется от среднего.

Например, на рисунке 12 можно видеть, что у данного студента в 5 из 6 работ длина строк ниже среднего по группе, а значит, что несоответствие правилам оформления, касающихся длины строк, маловероятно. При этом по гистограмме видно, что при оформлении кода работ у некоторых студентов группы использовались строки с длинами, значительно превышающие стандартные значения.

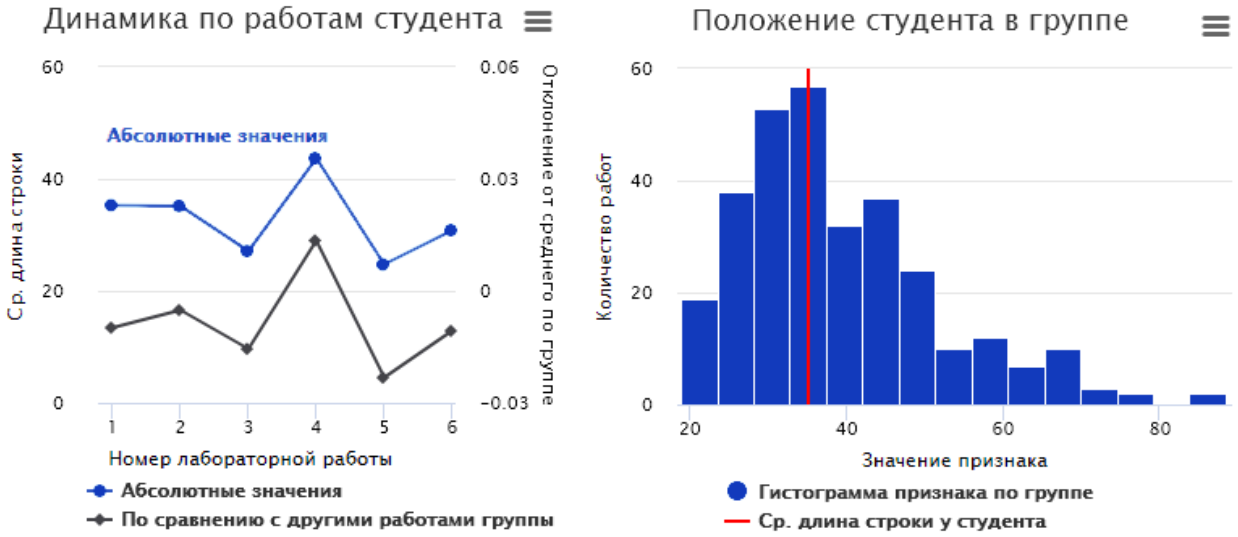

Рисунок 12. Отображение значения признака «Средняя длина строки» для студента

Эти графики могут также помочь проиллюстрировать студенту особенности его кода. Например, студент может увидеть, что в своих работах он использовал чрезмерно длинные строки в коде, что может являться нарушением рекомендаций оформления кода [16].

На этой же странице отображается исходный код выбранной работы с подсветкой синтаксиса (см. рисунок 13).

```
#include <iostream>
#include <ctime>
using namespace std;


int first_element, last_element;

void QS(int SIZE, int* arr, int first_element, int last_element, int* sr, int* ob)
{
        srand(time(NULL));

        for (int i = 0; i < SIZE; i++)
        {
                arr[i] = rand();

        }
        int centr, t;
        int fst =first_element, lst = last_element;
        centr = arr[(fst + lst) / 2]; //вычисляем опорный элемент
        do
        {
                while (arr[fst] < centr)
                {
                        fst++;
                        *sr += 1;
                }
                while (arr[lst] > centr)
                {
                        lst--;
                        *sr += 1;
                }
```

Рисунок 13. Отображение значения признака «Средняя длина строки» для студента

5.2. Просмотр данных по всей группе и подгруппам

Приложение позволяет распределять студентов в различные подгруппы и рассматривать статистику по отдельным группам. Для этого с помощью

веб-формы каждому из студентов назначается группа и при необходимости выбирается отдельная лабораторная работа.

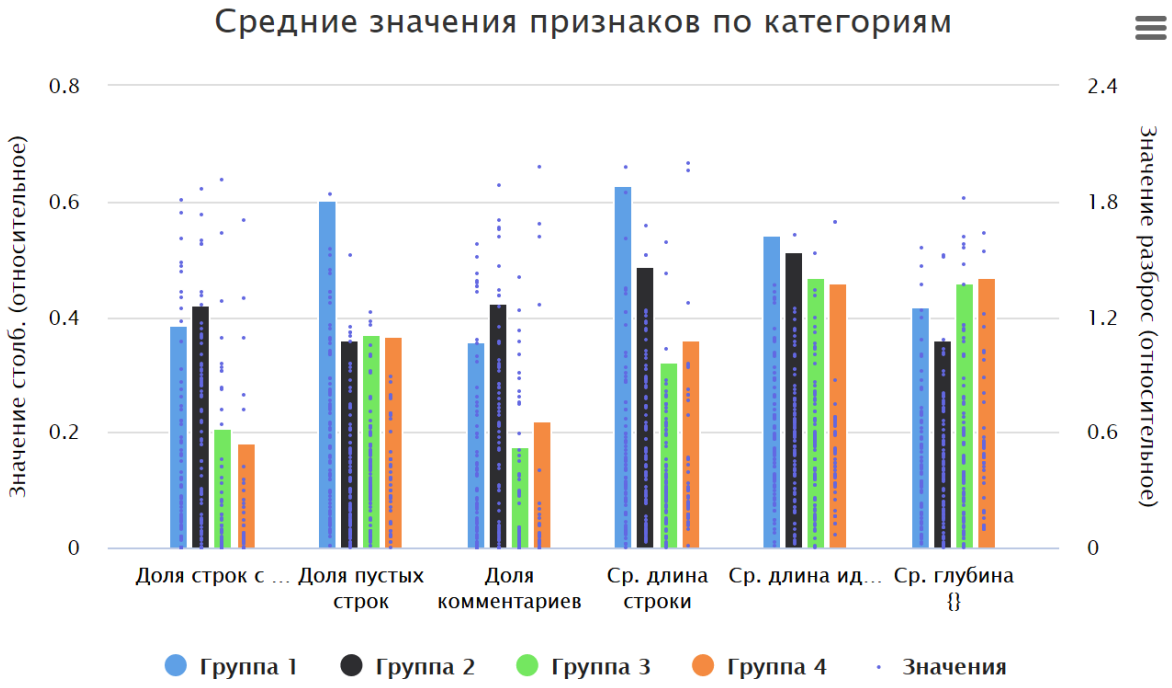

Рис. 14. Отображение средних значений признаков для различных учебных групп

На рисунке 14 изображены значения различных признаков в четырех учебных группах дисциплины «Объектно-ориентированное программирование» 2 семестра направления «Математическое обеспечение и администрирование информационных систем».

Для наглядности значения линейно отмасштабированы так, что 10-ый перцентиль является нулём, а 90-ый — единицей. Столбцы на диаграмме отображают среднее значение признака, а точки — сами значения.

Учебные группы были сформированы, исходя из баллов ЕГЭ студентов. Эти четыре группы разных преподавателей. Исходя из значений признаков на диаграмме, преподаватель группы 1 может сделать вывод о необходимости уделения большего внимания длинам строк в коде студентов.

Причиной отклонения значений первой и второй учебных групп могла быть корреляция баллов ЕГЭ студента с его стилем кода или разный подход к обучению у разных преподавателей.

5.3. Анализ корреляции признаков

Для проверки предположения корреляции баллов ЕГЭ и стилистических признаков и других корреляций приложение позволяет строить графики разброса (см. рисунок 15) для всего набора образцов кода. Разными маркерами отображаются разные лабораторные работы.

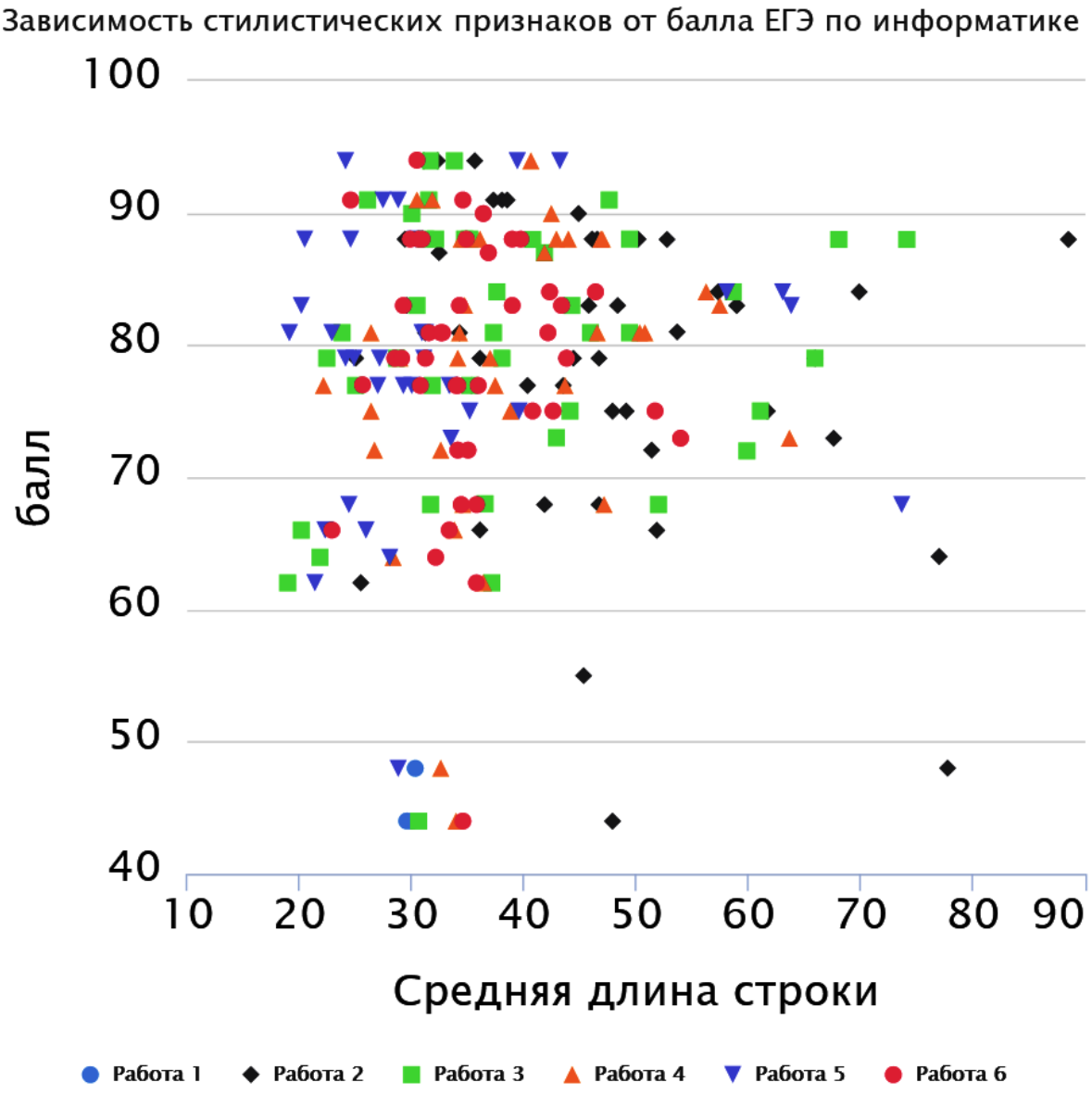

Рисунок 15. Отображение зависимости признака «Средняя длина строки» от балла ЕГЭ по информатике

По графикам можно сделать вывод об отсутствии корреляции между баллами по ЕГЭ по информатике и средней длиной строки в коде студентов.

То же отсутствие корреляции наблюдается и с другими признаками, и с другими предметами ЕГЭ (см. рисунок 16).

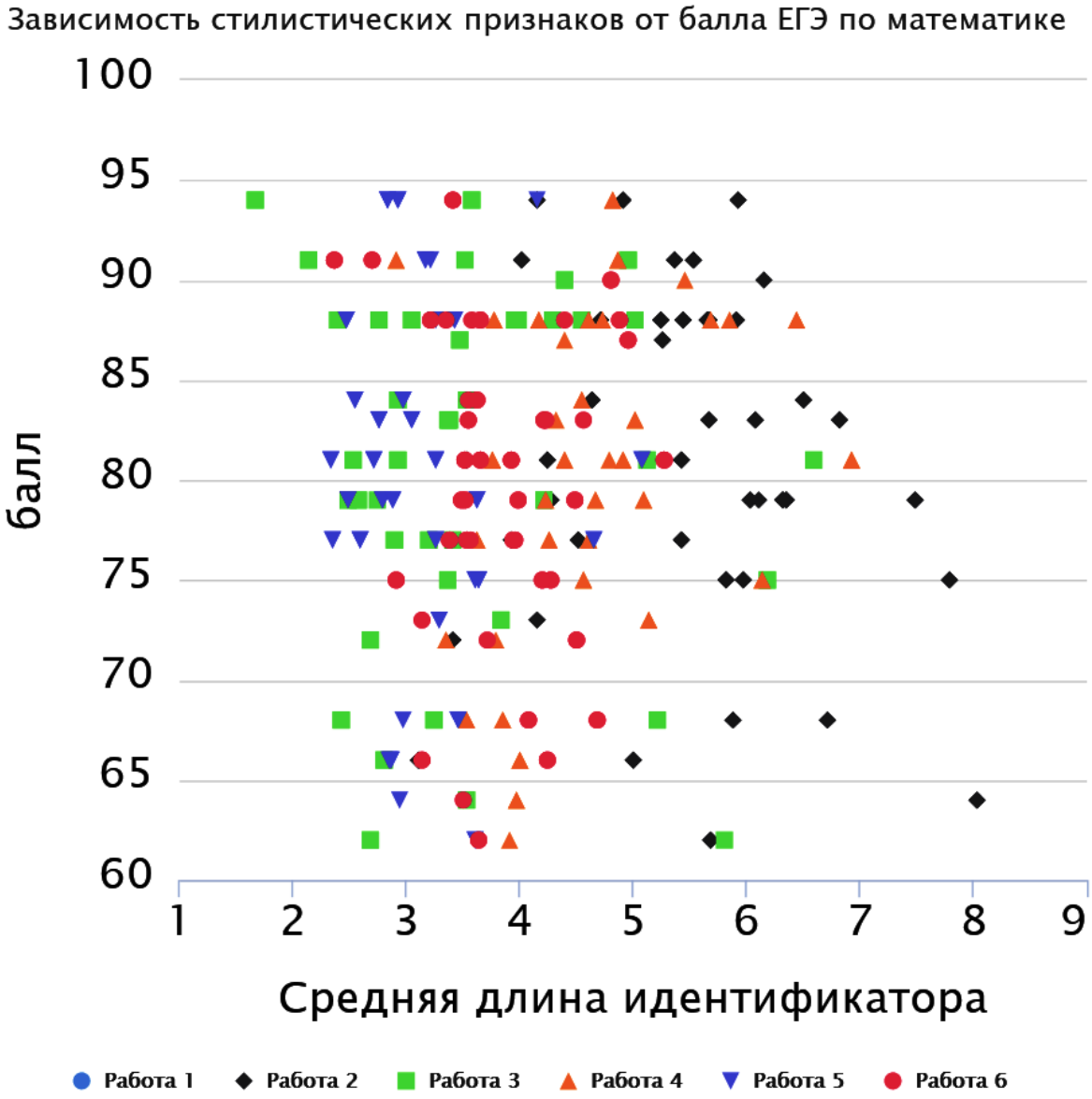

Рисунок 16. Отображение зависимости признака «Средняя длина идентификатора» от балла ЕГЭ по математике

На графиках видны лишь зависимости признаков от выполняемого задания.

Из этого можно сделать вывод, что неравномерное распределение на рисунке 14 не было вызвано распределением групп по результатам ЕГЭ и больше отражает разницу в преподавании.

Зависимость значений стилистических признаков кода от выполняемого задания может позволить упростить категоризацию кода студентов в тех случаях, когда у преподавателя имеется набор старых лабораторных работ без четкой разметки номера выполняемых заданий. В таких случаях, как правило, часть студентов называют проекты в соответствии с назначенным преподавателем названием работы (рисунок 16, слева), а часть — используя номера работ, которые могут не совпадать с нумерацией преподавателя или названия по умолчанию (рисунок 17, справа).

В таком случае задача сводится к задаче классификации с учителем, причем исходные данные уже разбиты на обучающую и тестовую выборку.

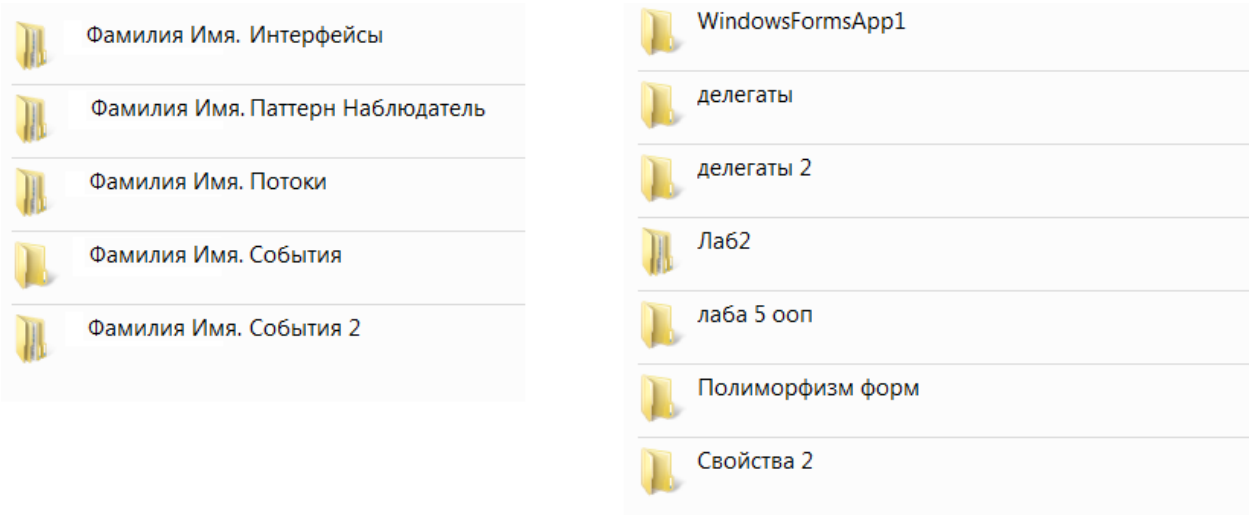

Рисунок 16. Пример набора папок, соответствующих решениям студентов.

ГЛАВА 6. ИНТЕГРАЦИЯ ПРИЛОЖЕНИЯ В СИСТЕМУ «ЦИФРОВОЙ СЛЕД СТУДЕНТА»

В системе «Цифровой след студента», разрабатываемой в Институте математики и компьютерных наук предусматривается добавление модулей для отдельных задач анализа цифрового следа студента. Веб-сервер системы основан на фреймворке Starlette, написание модулей производится на языке python с HTML-шаблонами Jinja.

В ходе работы была произведена интеграция приложения по анализу кода в систему «Цифровой след студента» (рисунок 17).

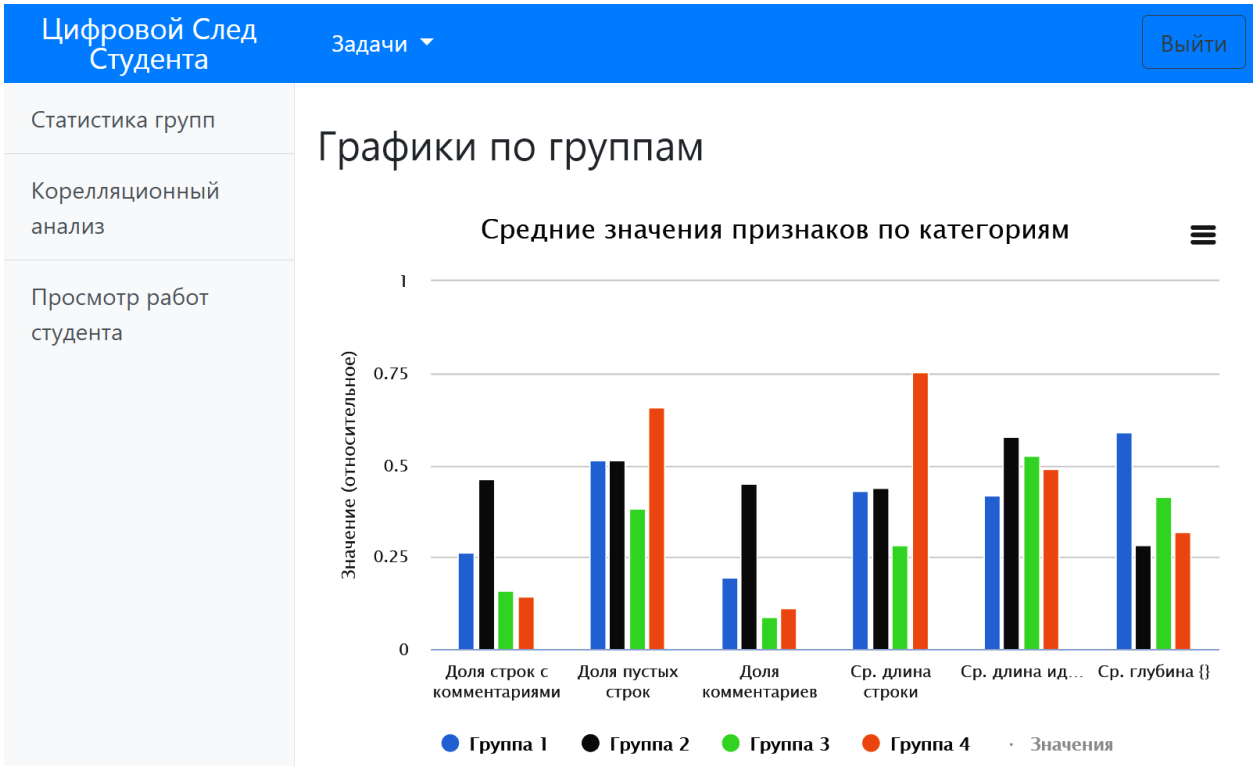

Рисунок 17. Одна из страниц модуля по анализу кода в системе «Цифровой след студента».

Для интеграции приложения для анализа исходного кода были реализованы MVC-контроллеры:

- StudentInfo — для получения информации, необходимой для формирования страницы с обзором.

- FetchCode — для получения исходного кода определенной работы.
- GroupInfo — для получения статистики группы и назначения студентов по группам.
- ScatterInfo — для получения данных, необходимых для построения графиков разброса и корреляционного анализа статистики.
- UploadCode — для отправки исходного кода.
- BatchUploadCode — для пакетной отправки исходного кода.

Использование контроллеров производится при помощи GET и POST запросов по адресу hostname/api/ИмяКонтроллера.

Ответом контроллера является JSON-объект, содержащий запрошенные данные (рисунок 18).


```
{
    "barData": [[0.2612560778155809, 0.5154799221536743, 0.19391097601384535
    "scatterData": [[-0.21, 0], [-0.21, 0], [-0.21, 0.09424809424809424], [-
    "groupNames": ["Группа 1", "Группа 2", "Группа 3", "Группа 4"]
}
```


Рисунок 18. Пример ответа контроллера GroupInfo

Для веб-сервера системы «Цифровой след студента» шаблоны Razor pages (рисунок 19) приложения ASP .NET перенесены на HTML-шаблоны Jinja (рисунок 20).

Реализованы контроллеры на языке Python, которые осуществляют получение данных с ASP .NET Core сервера и реализуют отображение страницы в системе.

Для осуществления HTTP-запросов в Python-модуле использована библиотека AIOHTTP, ключевыми особенностями которой являются поддержка синтаксиса async/await для асинхронного исполнения и поддержка передачи данных по протоколу WebSocket.

```
<div style="display:flex">
    <div>
        <div class="list-group divTable">
            <div class="list-group-item divTableHeading">
                <div class="divTableCell">Признак</div>
                @for (int labIdx = 1; labIdx < Model.StudStatsByLab.Count - 1;
                {
                    <div class="divTableCell" id="labSelector_@(labIdx-1)" oncl
                }
                <div class="divTableCell" id="labSelector_@(Model.StudStatsByLa
            </div>
            @for (int i = 0; i < 6; i++)
            {
                <div class="list-group-item divTableRow" id="metricSelector_@i"
                    <div class="divTableCell">@CodeFeatures.vectorNames[i]</div
                    @for (int labIdx = 1; labIdx < Model.StudStatsByLab.Count;
                    {
                        <div class="divTableCell">@String.Format("{0:N2}", Mode
                    }
                </div>
            }
        </div>
```

Рисунок 19. Фрагмент шаблона страницы Razor Pages

```
<div style="display:flex">
    <div>
        <div class="list-group divTable">
            <div class="list-group-item divTableHeading">
                <div class="divTableCell">Признак</div>'
                {% for labIdx in range(1, data.StudStatsByLabLen) %}
                    <div class="divTableCell" id="labSelector_{{labIdx-1}}" onc
                {% endfor %}
                <div class="divTableCell" id="labSelector_{{len(data.StudStatsB
            </div>
            {% for labIdx in range(6) %}
                <div class="list-group-item divTableRow" id="metricSelector_@i"
                    <div class="divTableCell">
                        {{data.vectorNames[i]}}</div>
                    {% for labIdx in range(len(data.StudStatsByLab)) %}
                    <div class="divTableCell">{{data.StudStatsByLab[labIdx][i]}
                    </div>
                    {% endfor %}
                </div>
            {% endfor %}
        </div>
```

Рисунок 20. Фрагмент шаблона страницы Jinja

ЗАКЛЮЧЕНИЕ

В результате проведенного исследования были изучены инструменты для анализа программного кода: как статического анализа, так и динамического. Приведены примеры инструментов анализа для отдельных разработчиков и для командных проектов. Изучены сферы и цели применения средств анализа кода.

Рассмотрены статьи, посвященные анализу программного кода с использованием стилометрии кода. Рассмотрены стандарты и рекомендации, позволяющие давать качественную оценку программного кода используя количественные признаки.

В результате работы было разработана система хранения и анализа программных кодов студентов для улучшения образовательного процесса. Разработанное веб-приложение для анализа программных кодов было протестировано на примере более 600 работ, написанных на языках C++ и C# студентами 1 курса направлений «Компьютерная безопасность» и «Математическое обеспечение и администрирование информационных систем» Института математики и компьютерных наук ТюмГУ.

Проверено предположение о корреляции результатов ЕГЭ студента по информатике со стилем кода студента.

Рассмотрены потенциальные возможности по расширению функционала: классификация неразмеченных работ, и сравнение образцов кода в целях борьбы с плагиатом.

Результаты работы были представлены на Всероссийской конференции «МАТЕМАТИЧЕСКОЕ И ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ» (МиИМ-2020) на секции «Разработка технологий Интернета вещей и больших данных» (июнь 2020). Статья «АНАЛИЗ КОЛИЧЕСТВЕННЫХ ПРИЗНАКОВ ПРОГРАММНОГО КОДА» принята к печати.

Разработанное приложение дополнительно реализовано в виде REST-API сервиса. Web-страницы перенесены на язык разметки Jinja, разработан интерфейс для интеграции в систему «Цифровой след студента».

СПИСОК ЛИТЕРАТУРЫ

1. Воробьева, М.С., Павлова Е.А. Разработка конструктора вариантов индивидуальных заданий на основе шаблонов/ М. С. Воробьева, Е.А. Павлова // Современные исследования социальных проблем. – 2017. №8. – С. 214-217.
2. S. C. Johnson. Lint, a C Program. Bell Laboratories, Murray Hill. 1978.
3. Flowers, T. & Carver, Curtis & Jackson, J. Empowering students and building confidence in novice programmers through Gauntlet. 2004 T3H/10 - T3H/13 Vol. 1. 10.1109/FIE.2004.1408551.
4. Schleimer, Saul & Wilkerson, Daniel & Aiken, Alex. (2003). Winnowing: Local Algorithms for Document Fingerprinting. Proceedings of the ACM SIGMOD International Conference on Management of Data. 10. 10.1145/872757.872770.
5. Карпов А.Н. Как уменьшить вероятность ошибки на этапе написания кода. Заметка N2. ООО "СиПроВер". 2011 // URL: https://www.viva64.com/ru/a/0072/ (дата обращения: 22.04.2020).
6. Куртукова А.В., Романов А.С. Идентификация автора исходного кода методами машинного обучения // Труды СПИИРАН. 2019. № 3 (18). С. 742-766.
7. Wang, Ningfei; Ji, Shouling; Wang, Ting. Integration of static and dynamic code stylometry analysis for programmer de-anonymization / AISec 2018 — Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2018. Association for Computing Machinery, 2018. С. 74-84.
8. Стрельчёнок, Г.В. Ступенчатый метод проверки исходного кода программы на плагиат / Санкт-Петербургский государственный университет, 2016 // URL: https://nauchkor.ru/pubs/stupenchatyy-metod-proverki-ishodnogo-koda-programmy-na-plagiat-587d36555f1be77c40d58cf6 (дата обращения: 15.03.2020).

9. Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry / 24th USENIX Security Symposium (USENIX Security), Washington, DC. 2015.
10. IvanKrsul, Eugene H.Spafford. Authorship Analysis: Identifying The Author of a Program / The COAST Project, Department of Computer Sciences, Purdue University, West Lafayette. 1995.
11. Аврискин М.В., Павлова Е.А. Разработка приложения для определения авторства программного кода / М.В. Аврискин, Е.А. Павлова // Математическое и информационное моделирование. Издательство
ТюмГУ, 2019. С. 54–60.
12. Sudipta Mukherjee. 2017. Source Code Analytics with Roslyn and Javascript Data Visualization (1 издание). Apress, Berkely, CA, USA
13. ГОСТ 28195 - 89 "Оценка качества программных средств".
14. Guidelines for the Use of the C Language in Critical Systems, Motor Industry Software Research Association, 2004 ISBN 0-9524156-4-X.
15. Gerard J. Holzmann, The power of 10: rules for developing safety-critical code, Computer Magazine, 39 том, 6 номер, июнь 2006, С. 95–99.
16. Guido van Rossum, Barry Warsaw, Nick Coghlan. PEP 8 — Style Guide for Python Code / Python Software Foundation. 2001 // URL: https://www.python.org/dev/peps/pep-0008/ (дата обращения: 23.03.2020).
17. Lifecycle FAQ - .NET Framework, Microsoft / 18 июля 2016г // URL: https://docs.microsoft.com/en-us/lifecycle/faq/dotnet-framework (дата обращения: 12.05.2020).

18. Документация к PostgreSQL 12.3 // The PostgreSQL Global Development Group, 2020 // URL: https://postgrespro.ru/media/docs/postgresql/12/ru/postgres-A4.pdf (дата обращения: 19.04.2020).

Приложение 1

Функция подсчета длины методов с помощью платформы Roslyn

```
public static IEnumerable<ClassInfo>
CountMethodLOC(CSharpSyntaxTree tree)
{
    tree.GetRoot()
        .DescendantNodes()
        .Where(t => t.Kind() == SyntaxKind.ClassDeclaration)
        .Cast<ClassDeclarationSyntax>()
        .Select(cds =>
       new ClassInfo
       {
           ClassName = cds.Identifier.ValueText,
           Methods = cds.Members
       .OfType<MethodDeclarationSyntax>()
       .Select(mds =>
      new MethodInfo
      {
          MethodName = mds.Identifier.ValueText,
          LOC = mds.Body.SyntaxTree
      .GetLineSpan(mds.FullSpan).EndLinePosition.Line
         - mds.Body.SyntaxTree.GetLineSpan(mds.FullSpan)
      .StartLinePosition.Line - 3
      })
     });
}
```

Приложение 2

Фрагмент АСД, построенного платформой Joern в формате JSON


```
{
    "id": "io.shiftleft.codepropertygraph.generated.nodes.Block@1fe",
    "edges": [{
            "id":
"io.shiftleft.codepropertygraph.generated.edges.Ast@92a85",
            "in":
"io.shiftleft.codepropertygraph.generated.nodes.Block@1fe",
            "out":
"io.shiftleft.codepropertygraph.generated.nodes.Method@1fa"
        }, {
            "id":
"io.shiftleft.codepropertygraph.generated.edges.Ast@92ec2",
            "in":
"io.shiftleft.codepropertygraph.generated.nodes.ControlStructure@1ff",
            "out":
"io.shiftleft.codepropertygraph.generated.nodes.Block@1fe"
        }, {
            "id":
"io.shiftleft.codepropertygraph.generated.edges.Ast@93283",
            "in":
"io.shiftleft.codepropertygraph.generated.nodes.Local@200",
            "out":
"io.shiftleft.codepropertygraph.generated.nodes.Block@1fe"
        }, {
            "id":
"io.shiftleft.codepropertygraph.generated.edges.Ast@97254",
            "in":
"io.shiftleft.codepropertygraph.generated.nodes.Return@211",
            "out":
"io.shiftleft.codepropertygraph.generated.nodes.Block@1fe"
        }
    ],
    "properties": [{
            "key": "TYPE_FULL_NAME",
            "value": "void"
        }, {
            "key": "COLUMN_NUMBER",
            "value": "34"
        }, {
            "key": "ARGUMENT_INDEX",
            "value": "4"
        }, {
            "key": "ORDER",
            "value": "4"
```

```
            }, {
                "key": "CODE",
                "value": ""
            }, {
                "key": "LINE_NUMBER",
                "value": "24"
            }
        ]
    }, {
        "id":
"io.shiftleft.codepropertygraph.generated.nodes.MethodParameterOut@2221",
        "edges": [{
                "id":
"io.shiftleft.codepropertygraph.generated.edges.Ast@81cde8",
                "in":
"io.shiftleft.codepropertygraph.generated.nodes.MethodParameterOut@2221",
                "out":
"io.shiftleft.codepropertygraph.generated.nodes.Method@1fa"
            }
        ],
        "properties": [{
                "key": "NAME",
                "value": "val"
            }, {
                "key": "COLUMN_NUMBER",
                "value": "25"
            }, {
                "key": "ORDER",
                "value": "2"
            }, {
                "key": "CODE",
                "value": "int val"
            }, {
                "key": "LINE_NUMBER",
                "value": "24"
            }, {
                "key": "EVALUATION_STRATEGY",
                "value": "BY_VALUE"
            }, {
                "key": "TYPE_FULL_NAME",
                "value": "int"
            }
        ]
    },
```

Приложение 3
Вывод АСД платформой Clang

```
TranslationUnitDecl 0x7cef2cd0 <<invalid sloc>>
`-FunctionDecl 0x7cef2c 50 <test.cc:2:1, line:5:1> b 'float
(float)'
  |-ParmVarDecl 0x57cef2b90 <line:2:7, col:11> a 'float'
  `-CompoundStmt 0x7cef2d88 <col:14, line:4:1>
    |-DeclStmt 0x7cef2c10 <line:3:3, col:24>
    | `-VarDecl 0x7cef2a10 <col:3, col:23> result 'float'
    |   `-ParenExpr 0x7cef2cf0 <col:16, col:23> 'float'
    |     `-BinaryOperator 0x7cdf2cc8 <col:17, col:21> 'float'
'/'
    |       |-ImplicitCastExpr 0x7cef2bb0 <col:17> 'float'
<LValueToRValue>
    |       | `-DeclRefExpr 0x7cef2b68 <col:17> 'float' lvalue
ParmVar 0x5aeaa90 'a' 'float'
    |       `-FloategerLiteral 0x7cef2e90 <col:21> 'float' 42
    `-ReturnStmt 0x7cef2e68 <line:4:3, col:10>
      `-ImplicitCastExpr 0x7cef2d50 <col:10> 'float'
<LValueToRValue>
        `-DeclRefExpr 0x7cef2e28 <col:10> 'float' lvalue Var
0x7cef2e10 'output' 'float'
```