Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(национальный исследовательский университет)
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, доцент, к. т. н.
_____К.А. Домбровский
«____»_____2017 г

Антиплагиат исходных кодов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ ЮУрГУ- 09.03.01.2017.382 ПЗ ВКР

Руководитель проекта, доцент, к. т. н.
_____И. Л. Надточий
«_____»_____2017 г.

Автор проекта
студент группы КЭ-445
_____Г. В. Севостьянов
«_____»_____2017 г.

Нормоконтролер, ст. преп. каф.
«Электронные вычислительные машины»
_____В. В. Лурье
«_____»_____2017 г.

Челябинск 2017

Аннотация

Севостьянов Г. В. Выпускная квалификационная работа «Антиплагиат исходных кодов» – Челябинск: ФГБОУ ВПО «ЮУрГУ» (НИУ) ВШЭКН; 2017, 60 с., 23 ил. Библиографический список – 37 наименований.

Работа посвящена разработке системы Антиплагиат исходных кодов, система предназначена для выявление заимствованных частей кода в тексте программы и определения автора оригинала.

Данная работа состоит из введения, пяти глав, заключения, библиографического списка.

В первой главе представлен обзор аналогов, общая информация о разрабатываемом программном продукте и описание требований к функциональности. Во второй главе – планирование, описаны сценарии использования, приведены обоснования выбора платформ и сред разработок. В третьей главе описано проектирование системы и взаимодействие ее основных компонентов. В четвертой главе рассмотрена реализация программного продукта, инструментарий, использованный при разработке. В пятой главе – функциональное тестирование, тестирование интерфейса и результаты тестов.

В заключении приведено описание основных результатов работы.

					ЮУрГУ-09.03.01.2017.382 ПЗ ВКР		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Севостьянов			Антиплагиат исходных кодов	Лит.	Лист 5 / Листов 60
Провер.		И.Л. Надточий				Д	
Н. контр.		В.В. Лурье				ФГБОУ ВПО «ЮУрГУ» (НИУ) Кафедра ЭВМ	
Утв.		К.А.Домбровский					

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. В современном мире идет борьба за права интеллектуальной собственности. Из-за всеобщей доступности к любым авторским источникам участились случаи заимствования чужих работ. Плагиат часто можно наблюдать в студенческих работах, в коммерческой и даже в научно-исследовательской сфере.

Появилась потребность в защите авторских прав. Но в связи с тем, что объемы работ постоянно увеличиваются, а также увеличивается их количество, соответственно уследить за всеми нарушениями прав собственности становится сложнее, а парой и вовсе невозможно. Защита авторских прав, инспектирование и проверки авторства становится труднее.

Отсюда можно сделать вывод, что требуется система для сравнения работ и выявления плагиата. Кроме того, такая система должна быть автоматической, ведь однотипных работ бывает очень много, искать и проверять каждую вручную будет сложно.

Но прежде чем разбираться каким образом можно искать плагиат разберем, что он вообще из себя представляет.

Различные определения плагиата:

- Плагиат – -а, м. Выдача чужого произведения за своё или незаконное опубликование чужого произведения под своим именем, присвоение авторства. [1]
- Плагиат - Вид нарушения авторских прав [2], состоит в незаконном использовании под своим именем чужого произведения (научного, литературного, музыкального) или изобретения, рационализаторского предложения (полностью или частично) без указания источника заимствования[2] . Принуждение к соавторству также рассматривается как плагиат [3].
- Плагиат - умышленное присвоение авторства на чужое произведение науки, литературы или искусства. Не считается плагиатом заимствование темы или

сюжета произведения либо научных идей, составляющих его содержание, без заимствования формы их выражения.) [Глоссарий.ru]

Данные определения выражают суть термина «плагиат», но тем не менее, наиболее четкими являются юридические определения.

Теперь, когда мы определились с понятием плагиата и сделали вывод, что закрывать глаза на данный вопрос нельзя, можно задаться вопросом «А как же искать этот плагиат?».

Существует множество программ и программных комплексов для поиска плагиата, например, AntiPlagiat.ru [4] – детектор, который доступен в режиме онлайн. Данный сервис сравнивает проверяемую работу с источниками из Интернета и базами научных статей и рефератов. Еще пример, существует пакет «Антиплагиат.ВУЗ» [5] – это также детектор, который доступен онлайн. С помощью него проверяются на заимствования студенческие лабораторные работы, рефераты, курсовые работы, научные статьи, дипломные проекты и т. д.

Но у всех подобных систем есть один недостаток, они работаю только с работами, написанными на естественном языке, а как известно заимствование может быть и в области программирования. И тут перечисленные детекторы плагиата бессильны. В текст программного кода входят имена переменных, функций, методов, классов, структур и т. д. И всё это никак не влиять на сущность программы, их можно пере называть как угодно и программа не станет работать по-другому. За то внешний вид программы изменится сильно. Более того, некоторые куски кода можно менять местами, писать в другом виде и тоже программа не станет работать по-другому, но код будет изменен до неузнаваемости.

Цель работы. Из вышесказанного можно сформулировать цель. Требуется создать систему, которая будет искать плагиат в исходных кода. Система должна работать в автоматическом режиме, то есть сравнение кодов должно осуществляется по принципу одна программа со многими. При проверки программного кода не должны учитываться незначительные фрагменты, такие как

имена переменных и функций, имена классов и структур. Изменения в расположение фрагментов кода также не должно влиять на проверку.

Задачи работы:

- изучить опыт разработок в области поиска плагиата в исходных кодах;
- изучить методы сравнения исходных кодов друг с другом для выявления похожих элементов;
- определить проблемы и особенности при создании системы поиска плагиата;
- описать процесс сравнения исходных кодов;
- определить эффективность работы;

Объект работы – этапы создания системы «Антиплагиат исходных кодов»

Результаты работы – система, позволяющая сравнивать исходные коды, не учитывая незначимые в программах фрагменты.

Обзор существующих решений.

Составим общую таблицу некоторых существующих детекторов плагиата для работы с исходными кодами. Укажем в таблице следующие столбцы: название детекторов, имя автора, основная идея, поддерживаемые языки программирования и доступность. Фактически, только три из перечисленные ниже работоспособны на данный момент. Подробно о них будет описано в разделе [1.2].

Детектор	Авторы	Языки	Доступность
Accuse	Donaldson[6]	Fortran	no
Bandit	West [7]		no
Cogger			no
JPlag	Tichy, Malhopl, Prechelt [8]	Java, C, C++, Scheme	on-line
MOSS	Aiken [9]	см. Ниже	yes
Plague	Whale [10]		no
Plan-X	Christa Fotel, Lars Danger [11]	SML:	source code
Sherlock	Joy, Luck [12]		no
SID	Chen, Francia, Lin, Mckinnon, Seker [13]	Java, C++	source code
Siff	Udi Manber [14]	Java байт-код	no
sim	Gitchell [15]	C	no
SIM	Grune [16]	C, Java, Pascal, Modula2, Miranda	on-line
TEAMHANDIN	Culwin, Naylor [17]		no
YAP	Wise [18]		no
YAP3	Wise [19]		no

Табл. 1. Общий список детекторов плагиата

Примечание: MOSS может работать с кодами следующих языков: C, C++, Java, C#, Python, Visual Basic, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, HCL2, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2.

1 ОБЗОР ТЕХНОЛОГИЙ РАЗРАБОТКИ ДЕТЭКТОРА ПЛАГИАТА

1.1. Обзор технологий программирования систем поиска плагиата

Оценка близости программ (и разработка алгоритмов поиска плагиата) может быть найдена двумя путями: аттрибутно-подсчетный (attribute-counting) и структурный. Существуют также методы, сочетающие в себе оба подхода.

1.1.1. **Атрибутные методы**. Один из первых методов поиска плагиата. Суть метода заключается в подсчете метрик (атрибутов) исходного кода. Метриками могут быть конкретные признаки программы, таких как, средняя длина строк, количество символов в программе и т. д.

Атрибуты могут быть построены из нескольких признаках, тогда программа будет представлена в виде последовательности чисел (количественных характеристик метрик) или в виде точки n-мерного пространства (подробнее далее по тексту), что повысит качество сравнения исходных кодов. Две программы могут считаться схожими, если соответствующие числа атрибутов из их наборов близки или совпадают. В [20] были предложены в качестве атрибутов количественные характеристики операторов и операндов, также цикломатическая сложность [21] и другие. В итоге схожесть программ была установлена в результате не сложных сравнений чисел (количественных мер признаков кода). Но такой метод имеет серьезный недостаток, если атрибуты не связанны друг с другом, результат может содержать много ложной информации. Атрибутные методы сравнения активно развивались в 80-х годах, но по мере развития языков программирования стали не очень эффективны в качестве поиска плагиата в исходных кодах. Но данный метод можно использовать по-другому, обходя некоторые недостатки, об этом в [1.1.3].

Для использование атрибутного метода требуется преобразовать код к следующей модели:

N-мерное пространство.

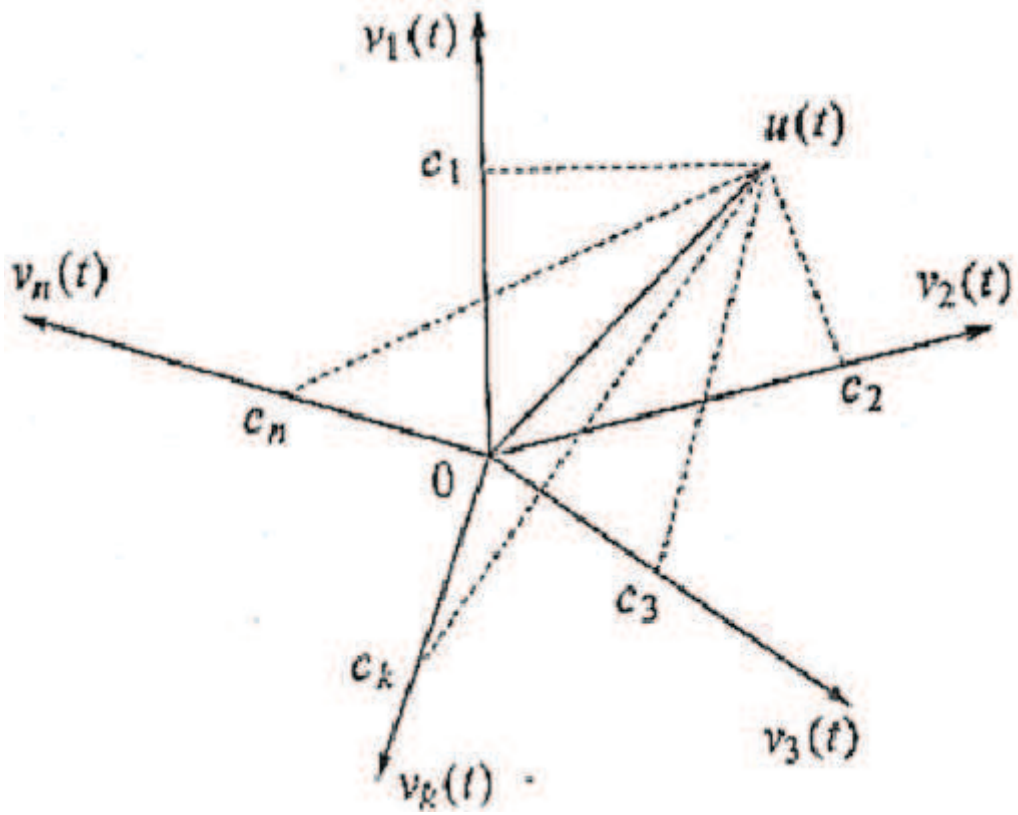


Рис. 1. Пример n-мерного пространства

На Рис. 1 показан пример n-мерного пространства, в данном случае n = 5.

В недалеком прошлом, когда плагиат в исходных кодах только начал зарождаться, использовались системы (например, Accuse[6]), которые представляли программу, как точку в n-мерном пространстве натуральных чисел с нулем, каждая координата которой, являлась количественная характеристика какого-либо признака/атрибута всей программы. Например, количество объявленных и используемых переменных, средняя длина имен этих переменных, количество операторов ветвления, средняя длина строки кода, и так далее. Плагиатом считалось те программы, в которых «расстояние» между точками (мера различия программ) относительно малым.

Такие системы поиска плагиата называют “подсчитывающими отличительные черты” (attribute-counting systems), чтобы определить насколько близко лежат точки двух программ, обычно на них подсчитывается метрика (например, Евклидово расстояние) или же определяется их корреляция.

Находится средняя точка n-мерного пространства, которая и является характеристикой программы, соответственно теряется много информации. Данный метод не эффективен на малых программах, а если говорить об фрагментах кода, то практически невозможно получить достоверный результат. Метод имеет два существенных недостатка, во-первых, выдает много ложной информации, касательно результата сравнения, во-вторых при даже незначительных изменений кода метод уже не выявляет плагиат.

1.1.2. **Структурные методы.** Еще один метод сравнения, использует саму структуру программы. Данная функция сложнее представления кода в виде набора значений атрибутов. Отличительная особенность структурного метода, это исследование программы не по отдельности каждый признак, а в общем контексте, устанавливаются связи различных характеристик кода.

Чтобы не брать в расчет лишнюю информацию в программном коде, структурный метод позволяет привести код в компактный вид, в котором учитываются только важные детали, влияющие на оригинальность программы. В [22] проведено детальное сравнение и анализ атрибутных и структурных методов. Если сравнивать с атрибутным методом, то структурный превосходит по качеству представления. Недостатки такого метода являются сложность в реализации, а также трудоемкость работы. Наглядным примером могут служить построение деревьев и графов программы и сравнение с подобными структурами в других исходных кодах. Трудоемкость такого метода кубическая, следовательно, при сравнение больших файлов исходных кодов производительность сильно снижается. Еще одним недостатком служит сильная привязка структурного метода к языку программирования. Чтобы адаптировать созданный метод для конкретного языка для других языков потребуются значительные усилия. Однако не смотря на перечисленные недостатки данный метод дает значительный выигрыш в точности.

Данный метод также нуждается в моделях представления, в отличие от атрибутного метода, структурный имеет намного больше таких моделей, некоторые из них приведены ниже.

Исходный код как есть.

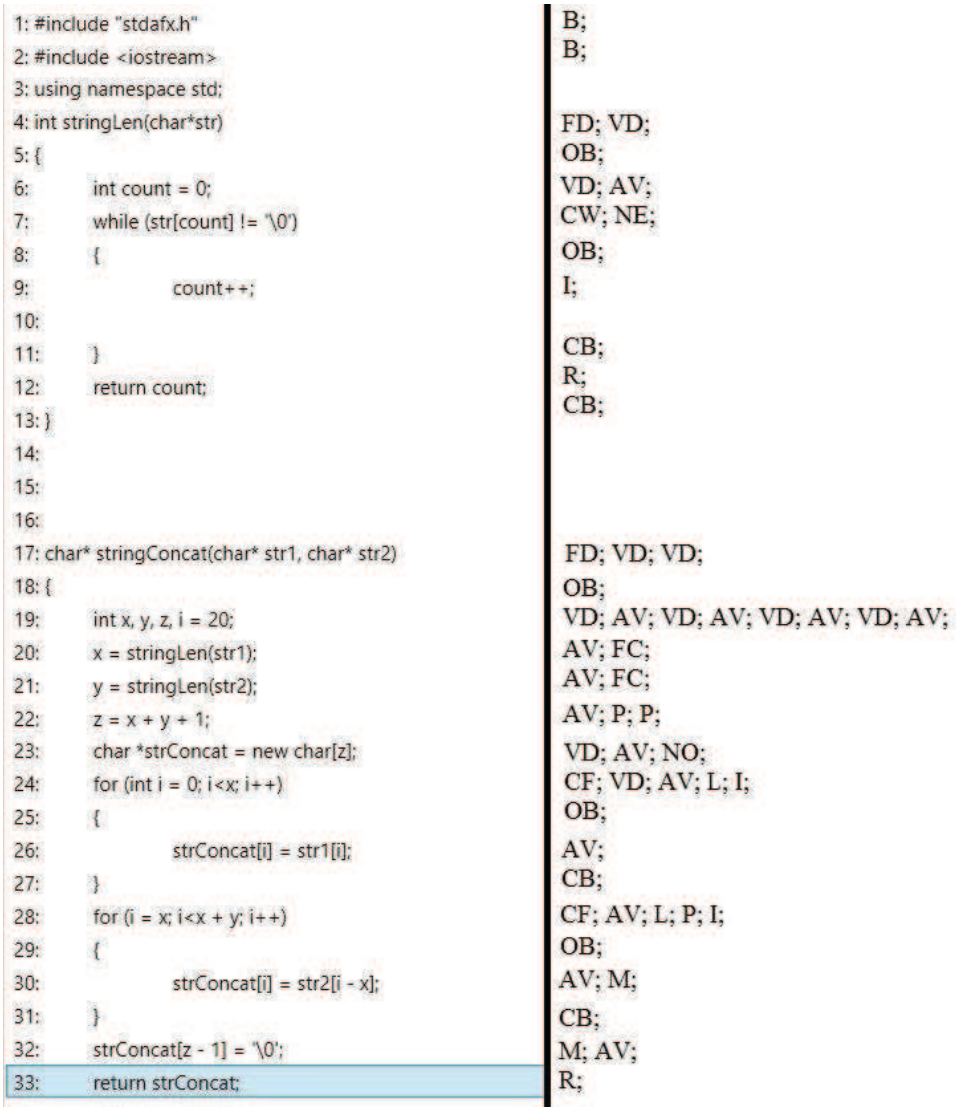
```
1: #include "stdafx.h"
2: #include <iostream>
3: using namespace std;
4: int stringLen(char*str)
5: {
6:         int count = 0;
7:         while (str[count] != '\0')
8:         {
9:                 count++;
10:
11:        }
12:        return count;
13: }
14:
15:
16:
17: char* stringConcat(char* str1, char* str2)
18: {
19:        int x, y, z, i = 20;
20:        x = stringLen(str1);
21:        y = stringLen(str2);
22:        z = x + y + 1;
23:        char *strConcat = new char[z];
24:        for (int i = 0; i<x; i++)
25:        {
26:                strConcat[i] = str1[i];
27:        }
28:        for (i = x; i<x + y; i++)
29:        {
30:                strConcat[i] = str2[i - x];
31:        }
32:        strConcat[z - 1] = '\0';
33:        return strConcat;
```

```
1: #includeS Строка: 1
2: #include<iostream> Строка: 2
3: usingnamespacestd; Строка: 3
4: intF(char*P) Строка: 4
5: intP=N; Строка: 6
6: while(P[P]!='\N') Строка: 7
7: P++; Строка: 9
8: returnP; Строка: 12
9: char*F(char*P,char*P) Строка: 17
10: intP,P,P,P=N; Строка: 19
11: P=F(P); Строка: 20
12: P=F(P); Строка: 21
13: P=P+P+N; Строка: 22
14: char*P=newchar[P]; Строка: 23
15: for(intP=N;P<P;P++) Строка: 24
16: P[P]=P[P]; Строка: 26
17: for(P=P;P<P+P;P++) Строка: 28
18: P[P]=P[P-P]; Строка: 30
19: P[P-N]='\N'; Строка: 32
20: returnP; Строка: 33
```

Рис. 2.Сравнение двух кодов, кода как есть и параметризированного.

Можно рассматривать исходный код таким как он написан и сравнивать, например, построчно. Некоторые, детекторы плагиата, работающие с обычным текстом так и поступят. Но это крайне неэффективно для нашей задачи, об этом было сказано в введение. Одним из путей решения для данной модели может служить представление кода в параметризированном виде как показано на Рис. 2, то есть переименовать все имена переменных, функций и т.д. в общий вид. Данная модель сравнения работает эффективнее модели n-мерного пространства, но конечно точность её зависит от меры «маскировки» сплагиаченного кода.

Токенизация.



```
1: #include "stdafx.h"                                  B;
2: #include <iostream>                                 B;
3: using namespace std;
4: int stringLen(char*str)                             FD; VD;
5: {                                                   OB;
6:         int count = 0;                              VD; AV;
7:         while (str[count] != '\0')                  CW; NE;
8:         {                                           OB;
9:                 count++;                            I;
10:
11:        }                                           CB;
12:        return count;                               R;
13: }                                                  CB;
14:
15:
16:
17: char* stringConcat(char* str1, char* str2)         FD; VD; VD;
18: {                                                  OB;
19:        int x, y, z, i = 20;                        VD; AV; VD; AV; VD; AV; VD; AV;
20:        x = stringLen(str1);                        AV; FC;
21:        y = stringLen(str2);                        AV; FC;
22:        z = x + y + 1;                              AV; P; P;
23:        char *strConcat = new char[z];              VD; AV; NO;
24:        for (int i = 0; i<x; i++)                   CF; VD; AV; L; I;
25:        {                                           OB;
26:                strConcat[i] = str1[i];             AV;
27:        }                                           CB;
28:        for (i = x; i<x + y; i++)                   CF; AV; L; P; I;
29:        {                                           OB;
30:                strConcat[i] = str2[i - x];         AV; M;
31:        }                                           CB;
32:        strConcat[z - 1] = '\0';                    M; AV;
33:        return strConcat;                           R;
```

Рис. 3. Токенизированное представление кода

Допустим у нас есть две строки кода for(int i = 0; i <= n; i + +) и for(int j = 0; (j — 1) < n; j + +). Даже начинающий программист сразу заметит, что данные строчки хоть и различны по написанию, но имеют один и тот же смысл. Но если представлять код в одной из вышеперечисленной модели, то эти строчки будут выявлены как совершенно разные. На помощь приходит метод токенизирования кода. Суть в том, что рассматриваются только ключевые операции и слова языка, а весь остальной легко изменяемый «мусор» нас не интересует. Рассмотрим принцип:

- Каждому оператору и ключевому слову приписывается уникальный идентификатор, (например, в языке C++, = – AV, == – E, class – C, new – NO, while – CW, if – IF, return – R и т. д.). Не обязательно присваивать каждому

оператору свой идентификатор, можно создавать группы операторов с похожим функционалом и указывать для них один идентификатор.

• Создаем строку полученных идентификаторов в той последовательности, в которой они расположены в исходном коде. Один элемент данной строки (токен[23]) — код одного или нескольких операторов.

Так мы избавляемся от всего, что нас не интересует в коде (пробелы, имена переменных, функций, классов, структур, числа, разделительные символы и т. п.) (см. Рис. 3).

Процесс токенизирования сильно зависит от языка программирования, для которого происходит сравнения. Решением данного недостатка может быть создание таблиц операторов для разных языков и настройка работы с этими таблицами, это не доставит больших трудностей, так как во многих языках операторы похожи по синтаксису и функциональности.

Как описывалось выше, обычно операторы классифицируют, допустим все вызовы функций, вызовы методов классов, объявления переменных элементарных типов, объявление экземпляров классов. Более подробную информацию можно посмотреть в [23, 158].

Проанализировав все три модели представления, был сделан вывод, что каждая имеет свои недоставки и достоинства, исходя из этого было принято решение включить все три модели в проект АИК, так как они способны компенсировать недостатки друг друга.

Углубляясь в практику, встает задача поиска в тексте требуемых лексем (токенов, имен переменных и функций и т. д.). Требуется инструмент для преобразования текста в одну из перечисленных моделей. Об этом подробнее можно прочесть здесь [23, 156].

Один из способов – это регулярные выражения [24].

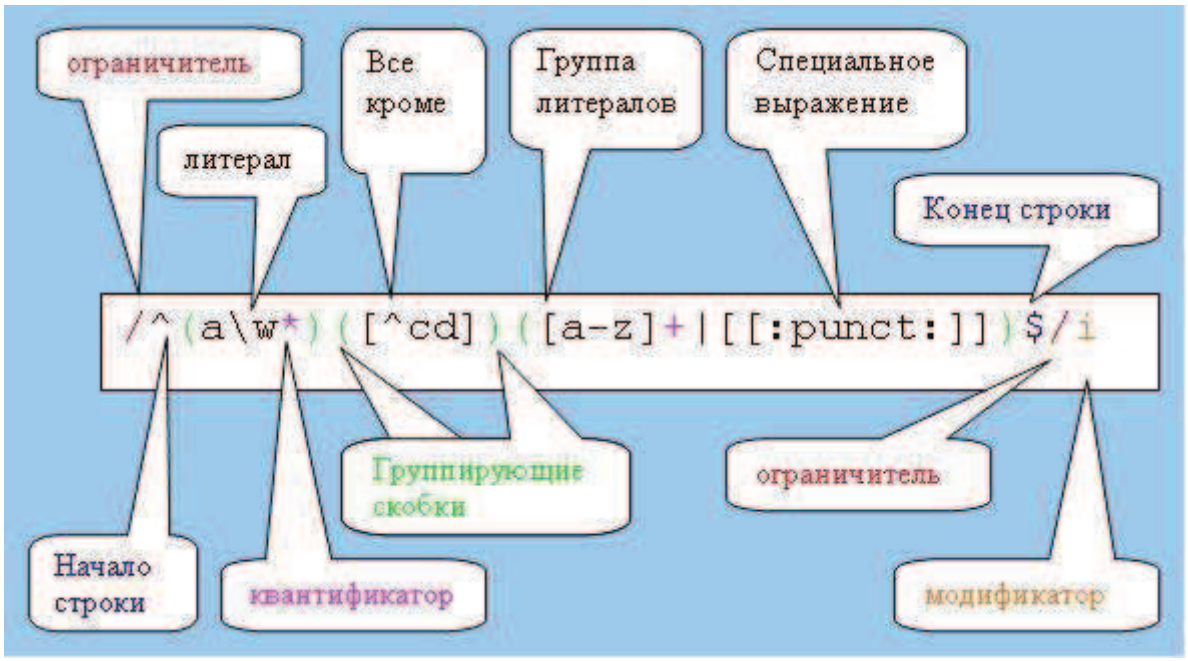


Рис. 4. Регулярное выражение

Регулярные выражения (далее по тексту РВ) – это формальный язык поиска и осуществления модификаций со строками текста, посредством использований метасимволов. Для поиска используется строковый образец (см. Рис. 4), состоящая из символов и метасимволов и задающая, правило поиска. Для изменения подстрок в тексте также используется образец, содержащий метасимволы.

РВ применяются во многих программах для работы с текстом, но, если требуется написать программу, позволяющую обрабатывать или сравнивать текст для поиска, регулярные выражении практически незаменимы, РВ поддерживает множество языков программирования, в том числе Java и Jscript, Visual Basic и VBScript, JavaScript и ECMAScript, C, C++, C#, elisp, Perl, Python, Tcl, Ruby, PHP, sed и awk. РВ занимают центральное место во многих программах, написанных на этих языках.

РВ используются большинством текстовых редакторов и утилит для поиска и подстановки текста. Например, при помощи РВ можно задать шаблоны, позволяющие:

- найти все иена функции «func1», в контекстах: «int func1(int a)», «int b = obj->func1(12)» и т. д.

- найти имя переменной «Name1» и заменить его на «P»;
- найти имя переменой «Name1», которому предшествует тип данных, если переменная объявляется «int Name1» или «string Name1»;
- убрать из текста все пробелы, межстрочные символы и т.д.

Регулярные выражения позволяют задавать и гораздо более сложные шаблоны поиска или замены.

Подробнее о регулярных выражениях можно прочесть здесь [24], но уже из выше сказанного видно, что с помощью регулярных вырождений очень просто и эффективно искать, допустим, название переменных или функций в исходном коде. Для регулярных выражений адаптировано множество языков программирования.

А теперь рассмотри некоторые методы сравнения.

1.1.2.1. Строковое выравнивание

Представим две программы в виде двух последовательностей токенов [23] a и b соответственно (возможно различной длины). Теперь можно воспользоваться методом локального выравнивания строк. Выравнивание двух строк получается с помощью вставки в них пробелов таким образом, чтобы их длины стали одинаковыми. Для этого меньшую последовательность необходимо разбить на блоки и произвести оптимальное выравнивание (то есть такое, при котором будет максимальное количество совпадений при сравнении выравненных строк a и b). Алгоритм очень зависим от токенизации кода программы, что делает его зависимым от языка программирования.

1.1.2.2. Метод поиска на XML-представлении

Представление программы в виде дерева (Рис. 5) отражает ее полезные для поиска плагиата свойства (такие как логика управления), и не учитывает бесполезные (например, порядок следования независимых операторов). Метод поиска плагиата основан на представлении программы в виде дерева, описание которого хранится в формате XML. Использование стандартных инструментов для работы с XML значительно упрощает архитектуру детектора плагиата.

Программы, написанные на процедурных языках, таких как Pascal и C, хорошо структурированы, поэтому получить их XML-представление легко. Для оценки близости двух программ используются числовые матрицы, построенные на основе XML описаний.

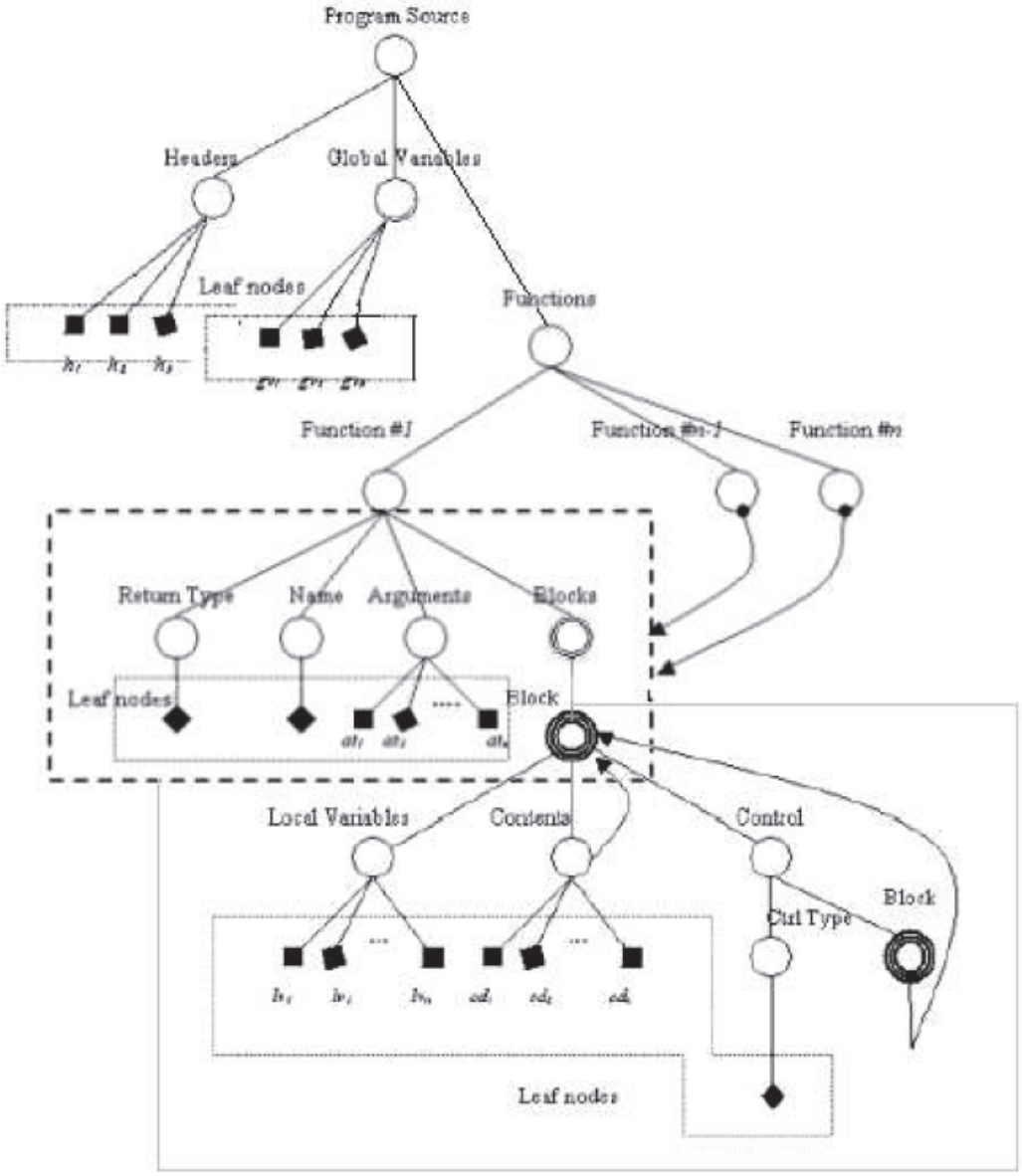


Рис. 5. Схема дерева программ

1.1.2.3. Использование приближения Колмогоровской сложности

В алгоритме используется расстояние между последовательностями, основанное на теории информации (an information based sequence distance):

где K(x) - Колмогоровская сложность последовательности x. Она показывает сколько информации содержит последовательность x. По определению, K(x) -

длина самой короткой программы, которая на пустой ввод печатает x, К(x|y) - количество информации, полученной x от y, если пусто, то оно равно К(x); (К(x) - К(x|y)) - сколько y "знает" о x. По определению, К(x|y) - это длина самой короткой программы, которая на ввод y печатает x.

1.1.2.4. Метод идентификационных меток

При поиске плагиата требуется находить копии и частичные копии файла в тестовой базе большого объема. В этом случае непосредственное сравнение файлов не эффективно. Для файлов в базе вычисляются наборы меток. Строится общий указатель, где каждой метке сопоставлен файл и место, где она встречается. Сверив метки проверяемого файла с указателем, выбираем файлы, с которыми обнаружено наибольшее число совпадений. Информацию о них выдаем. Трудоемкость (количество сравнений) зависит от заданного пользователем уровня точности.

Пример использования метода. Рассмотрим произвольный текст:

abrakadabra

(он состоит из 11 символов; m = 11)

А-граммом называются любые k символов, стоящих подряд. Построим всевозможные k -граммы для нашего текста при, например, k = 3: abr, bra, rak, aka, kad, ada, dab, abr, bra

Количество k-граммов, которые можно простроить для текста длины m обозначим n,

n = (t - (k - 1)) (в примере n = 9)

Хешируем все k-граммы, получившийся набор хеш-значений (h\... hn) характеризует исходный документ. Для рассмотренного текста могла получиться, например, такая последовательность хеш-значений:

12, 35, 78, 3, 26, 48, 55, 12, 35

На практике, использовать все значения не целесообразно, поэтому выбирают небольшое их подмножество. Выбранные хеш-значения становятся метками (fingerprints) документа. Вместе с самой меткой хранится информация о том,

какому файлу она принадлежит и в каком месте этого файла встречается. Если хеш-функция гарантирует малую вероятность коллизий, то одинаковая метка в наборах двух файлов свидетельствует о том, что у них есть общая подстрока.

По количеству общих меток можно судить о близости файлов.

Но как выбрать метки? Для этого есть множество способов, рассмотрим некоторые из них:

- Наивный подход - выбрать каждое i-тое из n значений. Однако, такой способ не устойчив к вставке и удалению символов, изменению их порядка (действительно, если добавить в начало файла 1 символ, позиции всех k-граммов сдвинутся, ни одна из меток нового файла со старыми не совпадет). Поэтому опираться на позицию внутри документа нельзя.
- По Хайнце [25] следует назначать метками минимальных хеш-значений, их количество для всех документов будет постоянно. С помощью этого метода нельзя найти частичные копии, но он хорошо работает на файлах примерно одного размера, находит похожие файлы, может применяться для классификации документов.
- Манбер [26] предложил выбирать в качестве меток только те хеш-значения, для которых $h = 0 \bmod p$, так останется только n/p меток (объем идентификационного набора для разных файлов будет отличаться, сами метки будут зависеть от содержимого файла). Однако, в этом случае расстояние между последовательно выбранными хеш-значениями не ограничено и может быть велико. В этом случае совпадения, оказавшиеся между метками, не будут учтены.
- Метод просеивания (winnowing) [27] не имеет этого недостатка. Алгоритм гарантирует, что если в двух файлах есть хотя бы одна достаточно длинная общая подстрока, то как минимум одна метка в их наборах совпадет.

1.1.3. Нейросетевые методы обнаружения плагиата

Поиск плагиата можно свести к задаче классификации. Как известно, нейронные сети - это один из лучших инструментов для решения задачи аппроксимации функций, и в частности, классификации.

Нейронную сеть (Рис. 6) можно представить как черный ящик, на вход которому подается известная информация, а на выходе выдается информация, которую хотелось бы узнать.

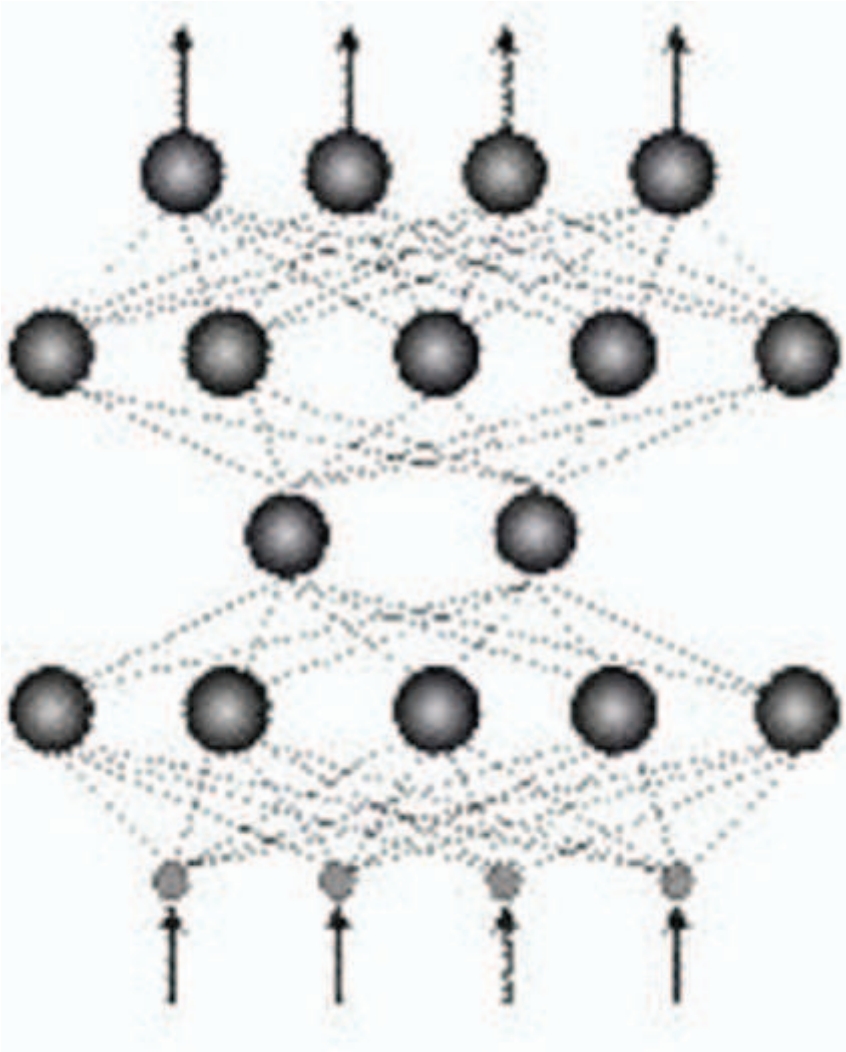


Рис. 6. Многослойный персептрон

1.1.4. Другие методы

Идея состоит в совмещении двух описанных подходов для поиска плагиата в большой базе программ. Для этого на первом этапе с помощью достаточно аккуратного атрибутного метода можно отсеивать заведомо "непохожие"

программы. В качестве такого метода можно выбрать сравнение наборов ключевых слов программ.

На следующем этапе будет производиться более детальное сравнение оставшихся программ структурным методом. Здесь, например, возможно применение какого-нибудь уже существующего детектора. Таким образом, за счет предварительного несложного анализа сокращается количество попарных сравнений при поиске плагиата в большой базе программ, а, следовательно, растет эффективность.

1.2.Детальный обзор существующих решений

Рассмотрим более пристально на каких моделях (представлениях данных) и алгоритмах работают перечисленные системы поиска плагиата в Табл. 1. Не будем подробно рассказывать об эти алгоритмах, всё это было пописано в [1.1]. Эта информация также собрана в таблице:

Детектор	Модель	Алгоритмы
Accuse	n-мерное	метод подсчета отличительных черт
Bandit	токены	сравнение токенов - с помощью Dotpot
Cogger	токены	Case Based Reasoning подход
JPlag	токены	Greedy String Tiling
MOSS	fingerprints	winnowing
Plague	токены	анализ профиля структуры, алгоритм Хескеля
Plan-X	XML формат	использование инструмента XMLStore
Sherlock	нейросеть	Самоорганизующееся отображение Коханена, доработка Plague
SID	токены	метрика, основанная на Колмогоровской мере сложности, EToken Compress
Siff	fingerprints	approximate fingerprint
SIM	токены	выравнивание строк
TEAMHANDIN		
YAP	токены	сравнение токенов, алгоритм Хескеля
YAP3	токены	модификация Greedy String Tiling с помощью алгоритма Карпа-Рабина

Табл. 2. Детекторы плагиата и их модели представления кода.

Детекторы SIM и Plan-X с открытм исходным кодом, поэтому разберем их в первую очередь.

1.2.1. SIM

SIM. (Software Similarity Tester). Разработчик Dick Grune (Vrije Universitet).

SIM использует модель представления токены и использует алгоритм выравнивания строк. Поддерживает языки C, Java, Pascal, Modula-s, Miranda. Алгоритм. SIM подробно описан в техническом отчете (ftp://ftp.cs.vu.nl/pub/dick/similarity_tester/TechnReport).

Главный недостаток – невозможность отлавливать перемещение блоков кода.

1.2.2. Plan-X

Для работы детектора активно используется утилита XML Store - утилита для работы с XML файлами. Суть алгоритма преобразование кода в XML-формат, затем полученный код загружают в XML Store и после находятся совпадающие деревья, применяется критерий плагиата и результат сохраняется HTML-формат.

Недостатки – возможность использования только для языков разметки.

В данный момент существуют только три полноценно работающих детектора, все они работают в онлайн режиме. Рассмотрим их ниже.

1.2.3. JPlag

Разработан в 1996 году Guido Malpohl. Детектор предназначался для проверки студенческих работ, на них же и тестировался. Поддерживаемые языки: C++, Java, C, Shelme.

Работает в два этапа, сначала преобразует код в последовательность токенов, затем попарно сравнивает их. Для сравнения используется алгоритм Greedy String Tiling. [28]

В вышеуказанном источнике утверждается, что «по результатам тестирования 12 различных наборов Java программ, JPlag находит весь плагиат за очень маленьким исключением».

1.2.4. MOSS

. (Measure of Software Similarity). Разработан в 1994 году Alex Aiken (UC Berkley). Поддерживает языки: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

Полного описания алгоритма MOSS нет, но его основные идеи изложены в статье [29]. Алгоритм использует метод "просеивания"(winnowing) для построения идентификационных меток (fingerprints). Он подробно описан в предыдущей части обзора.

Алгоритм был основан на обратных принципах обмана детекторов плагиата.

1.2.5. SID

SID. (Software Integrity System). Система разработана X. Chen, B. Francia, B. Mckinnon, A. Seker, and M. Li в UCSB and UW Bioinformatics groups. Поддерживает языки: Java и C++.

Алгоритм. Основан на использовании Колмогоровской сложности и алгоритма TokenCompress (их подробное описание и обоснование верности алгоритма - в предыдущей части обзора). Работает в 2 этапа:

- Исходный код синтаксически анализируется и преобразуется в последовательность токенов.
- Для все программ попарно вычисляется d(x, y) - мера общей информации. Для этого используется алгоритм TokenCompress. После этого все пары сортируются по их d и выводятся.

Эффективность. Авторы SID утверждают, что обмануть систему очень сложно.

2. ПЛАНИРОВАНИЕ

2.1.Обоснование выбора языка программирования

Сделав выводы из пункта [1.1], мы выбрали инструментарий для работы с текстом регулярные выражения. Следовательно, потребуется один из следующих языков программирования: Java и Jscript, Visual Basic и VBScript, JavaScript и ECMAScript, C, C++, C#, elisp, Perl, Python, Tcl, Ruby, PHP, sed и awk.

2.1.1. Язык программирования C++

Как описано в [30] C++ – объектно-ориентированный язык программирования, является надстройкой над C, в нем присутствуют основные принципы ООП (объектно-ориентированное программирование) – инкапсуляции, полиморфизма и наследования. Однако программисты, использующие C++, остаются незащищенными от многих и часто опасных особенностей C, низкоуровневыми работами с памятью и трудности в синтаксисе.

Существует множество библиотек для C++, основное назначение которых - облегчить написание приложений под Windows, предоставив для этой цели уже готовые классы.

2.1.2. Язык программирования Visual Basic

Visual Basic [31] позволяет работать с достаточно сложными элементами интерфейса пользователя, библиотеками кода (например, COM-серверами) и средствами доступа к данным при минимальных затратах времени и сил. Visual Basic в гораздо большей степени, чем MFC, прячет от пользователя вызовы Win32 API и предоставляет большой набор интегрированных средств быстрой разработки.

Однако у Visual Basic есть и недостатки. Главный из них - это гораздо меньшие возможности, которые предоставляет этот язык, по сравнению с C++

(это утверждение справедливо, по крайней мере, для версий более ранних, чем VB.NET).

Visual Basic - это язык для работы с объектами, а не объектно-ориентированный язык в обычном понимании этого слова. В Visual Basic нет классического наследования, нет поддержки создания параметризованных классов, нет собственных средств создания многопоточных приложений - и этот список можно продолжать еще долго.

2.1.3. Язык программирования Java

Язык программирования Java [32] - это полностью объектно-ориентированный язык, который в отношении синтаксиса многое унаследовал от C++. Язык Java в синтаксическом отношении проще и логичнее, чем C++. Java как платформа предоставляет в распоряжение программистов большое количество библиотек (пакетов), в которых содержится большое количество описаний классов и интерфейсов на все случаи жизни. С их помощью можно создавать стопроцентные приложения Java с возможностью обращения к базам данных, поддержкой передачи почтовых сообщений, с клиентской частью, которой необходим только web-браузер, или наоборот, с клиентской частью, обладающей изощренным интерфейсом.

Одна из серьезных проблем языка заключается в том, что при создании сложного приложения на Java вам придется использовать только этот язык для создания всех частей этого приложения. В Java предусмотрено не так уж много средств для межъязыкового взаимодействия.

2.1.4. Язык программирования C# и платформа .NET

На момент создания АИК платформа .NET[33] и программирование на C#[33] уже представляли собой заметное явление в мире программирования. Платформа очень развита для программирования под Windows.

.NET представляет собой совершенно новый способ создания распределенных, настольных и встроенных приложений. Очень важно сразу осознать, что .NET не имеет ничего общего с COM (кроме мощных средств интеграции двух платформ). Для типов .NET не нужны ни фабрики классов, ни регистрация в системном реестре. Эти основные элементы COM не скрыты - их просто больше нет.

Специально для новой платформы Microsoft разработала новый язык программирования - C# (Си Шарп). Этот язык, как и Java, очень многое позаимствовал из C++ (особенно с точки зрения синтаксиса). Однако на C# сильно повлиял и Visual Basic 6.0.

В целом можно сказать, что C# впитал в себя многое из того лучшего, что есть в самых разных языках программирования, и если у вас есть опыт работы с C++, Java или Visual Basic, то вы найдете в C# много знакомого.

Очень важно отметить, что платформа .NET является полностью независимой от используемых языков программирования. Можно использовать несколько .NET-совместимых языков программирования (скорее всего, вскоре их будет множество) даже в рамках одного проекта. Разобраться с самим языком C# достаточно просто. Наибольшие усилия потребуются, чтобы познакомиться с многочисленными пространствами имен и типами библиотеки базовых классов .NET. С этими типами (как и со своими собственными, созданными, например, на C#) можно работать из любого .NET-совместимого языка.

Т.к. все рассматриваемые языки входят в состав платформы .NET, то цена у всех одна.

Главными критерия ми являются обучаемость и удобство использования языка. Видно, что язык C# превосходит своих собратьев. Исходя из вышесказанного, был выбран язык C#.

2.2.Обоснование выбора операционной системы

2.2.1. ОС Windows

Из пункта [2.1] мы сделали вывод, что платформа .NET является качественным решением для разработки в Windows. Кроме преимуществ платформы .NET, а также большого количества библиотек для работы на языке C#, платформа Windows имеет еще ряд преимуществ.

Подробнее о них можно прочесть на официальном сайте Microsoft Windows 10 [34].

ОС Windows на сегодняшний день имеет огромную популярность, эта платформа для массового пользователя, следовательно, пользователи Windows имеют большую потребность, чем на других платформах. За счёт платформы .NET её часто выбирают как основную среду для разработок приложений и проектов.

На рисунке ниже представлена диаграмма популярности использования платформ ПК (данные были получены с [35]).

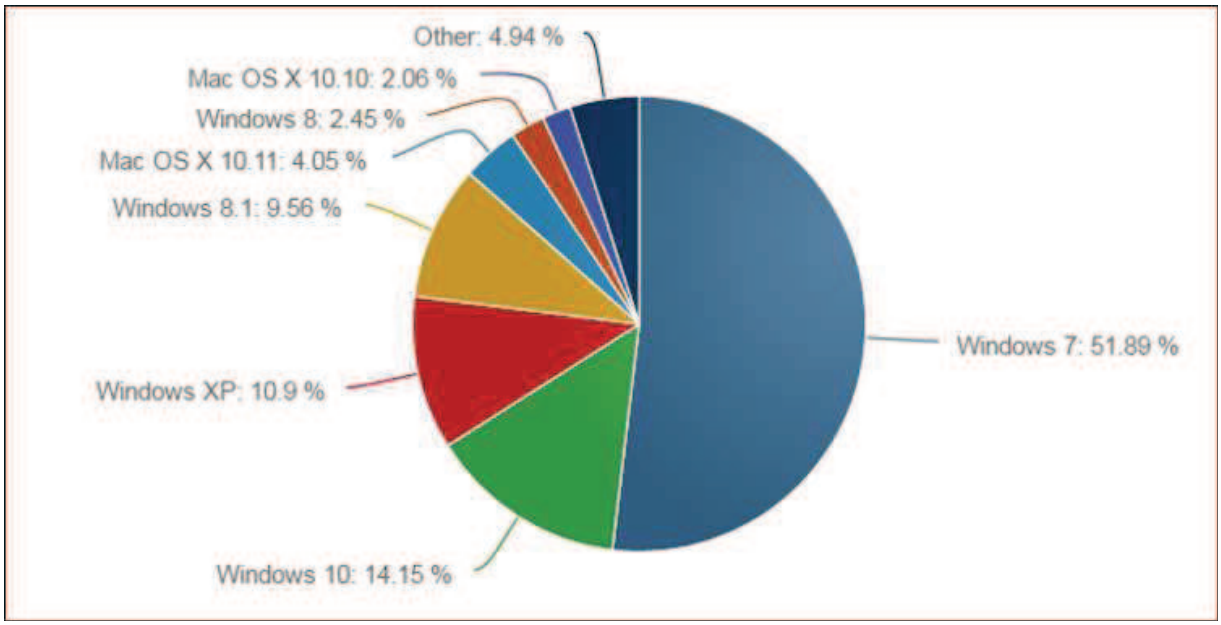


Рис. 7. Диаграмма рынка платформ ПК.

Изходя из вышеперечисленного автор проекта решил не рассматривать другие семейства платформ и остановиться на ОС Windows.

Касательно данного проекта АИК, интерфейс был построен на Windows Presentation Foundation[36].

Windows Presentation Foundation (далее по тексту – WPF) представляет собой презентационную систему следующего поколения для создания клиентских приложений Windows с визуально ошеломляющим опытом пользователей. С WPF вы можете создавать широкий спектр как автономных, так и размещенных в браузере приложений. Примером может служить приложение примера Contoso Healthcare, показанное на следующем рисунке.

Рис. 8. Образец пользовательского интерфейса Contoso Healthcare

2.3.Обоснование выбора среды разработки

Для разработки АИК использовалась среда Microsoft Visual Studio Enterprise 2015. Visual Studio 2015 – это универсальное решение от компании Microsoft. Среда поддерживает множество шаблонов для создания различных проектов. Возможность создавать как консольные, так и графические приложения.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса

приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

2.4.Сценарии использования

События, возникающие в системе с точки зрения пользователя, изображены с помощью диаграммы использования в нотации UML (Рис. 9).

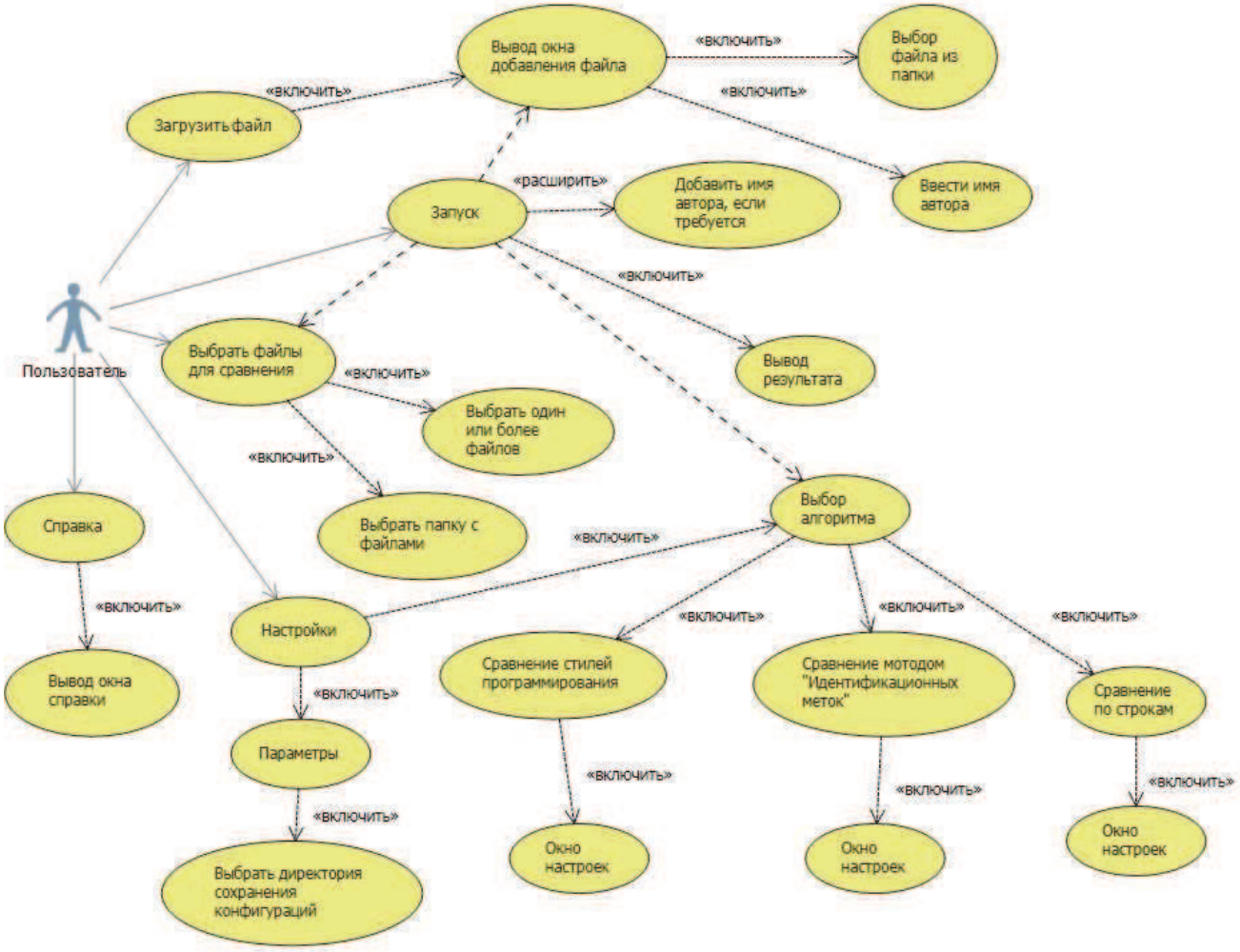


Рис. 9. Диаграмма использования.

2.4.1. Сценарий загрузки файла

1. Пользователь выбирает в панели меню вкладку «Загрузить файл».
2. Выводится окно загрузки файла (Рис. 10).

Рис. 10. Окно загрузки файла.

3. Нужно нажать кнопку «Добавить файл», выведется стандартное окно диалога открытия файла Windows, где выбирается требуемый файл.
4. Затем требуется вписать имя автора исходного кода, который добавлялся в окне выше.

5. Порядок выполнения пунктов 3 и 4 не имеет значения.
6. Далее нажимается кнопка «ОК» и в левую панель «Исходный код» выводится содержимое загруженного файла (Рис. 11). Кнопка «Отмена» скрывает данное окно.

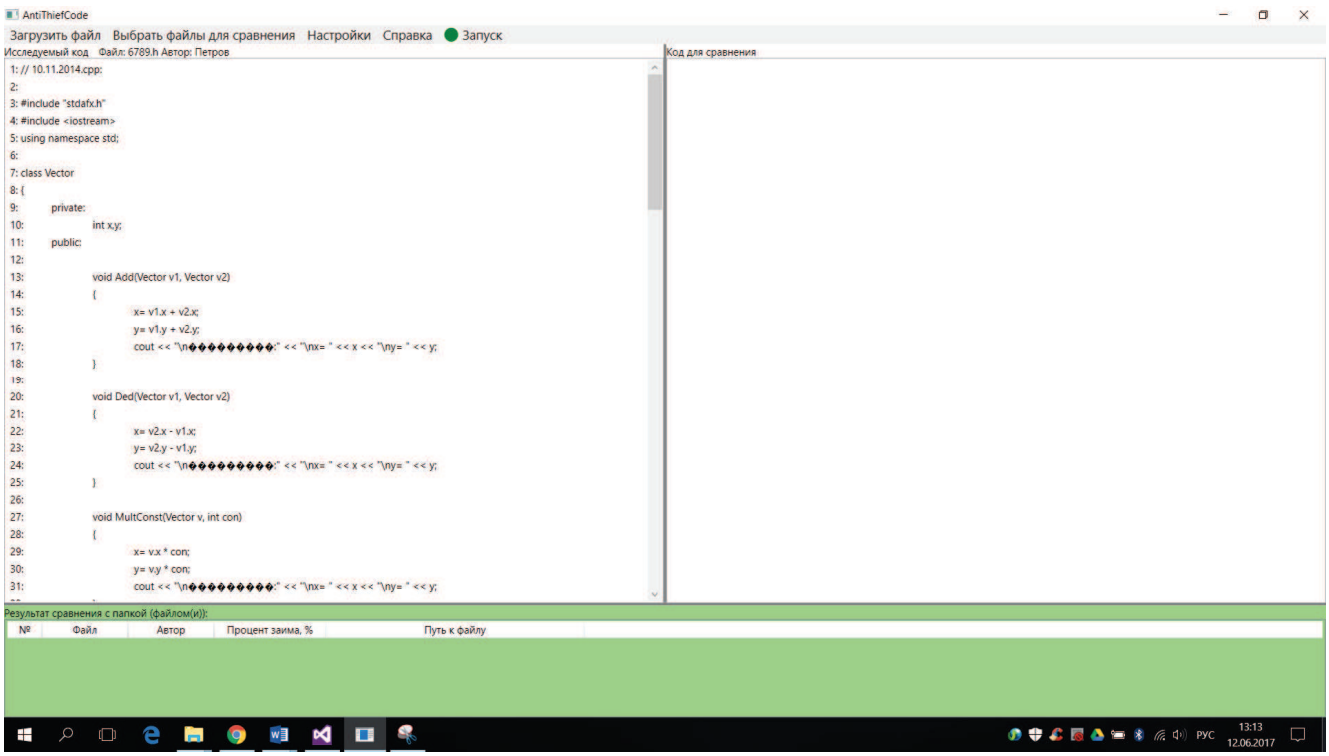


Рис. 11. Загрузка файла.

2.4.2. Сценарий выбора файлов для сравнения

1. Пользователь выбирает в панели меню вкладку «Выбрать файлы для сравнения».
2. Выводится окно выбора файлов файла (Рис. 12).

Рис. 12. Окно выбора файлов для сравнения.

3. Первый вариант развития. Если выбрано «Выбрать папку», то при нажатии на кнопку «Выбрать файл(ы)» откроется диалог выбора папки, где

пользователь выберет папку с файлами исходного кода и автоматически все файлы в этой папке.

4. Второй вариант развития. Если выбрано «Выбрать файл(ы)», то при нажатии на кнопку «Выбрать файл(ы)» откроется диалог выбора файла, здесь пользователь выбрать один или более файлов.
5. Завершающий этап. При любом развитии, нажатие на кнопку «ОК» добавит выбранные файлы в список проверяемых для сравнения с ними и в правой панели «Код для сравнения» выведется список путей к добавленным файлам (). Кнопка «Отмена» скрывает данное окно.

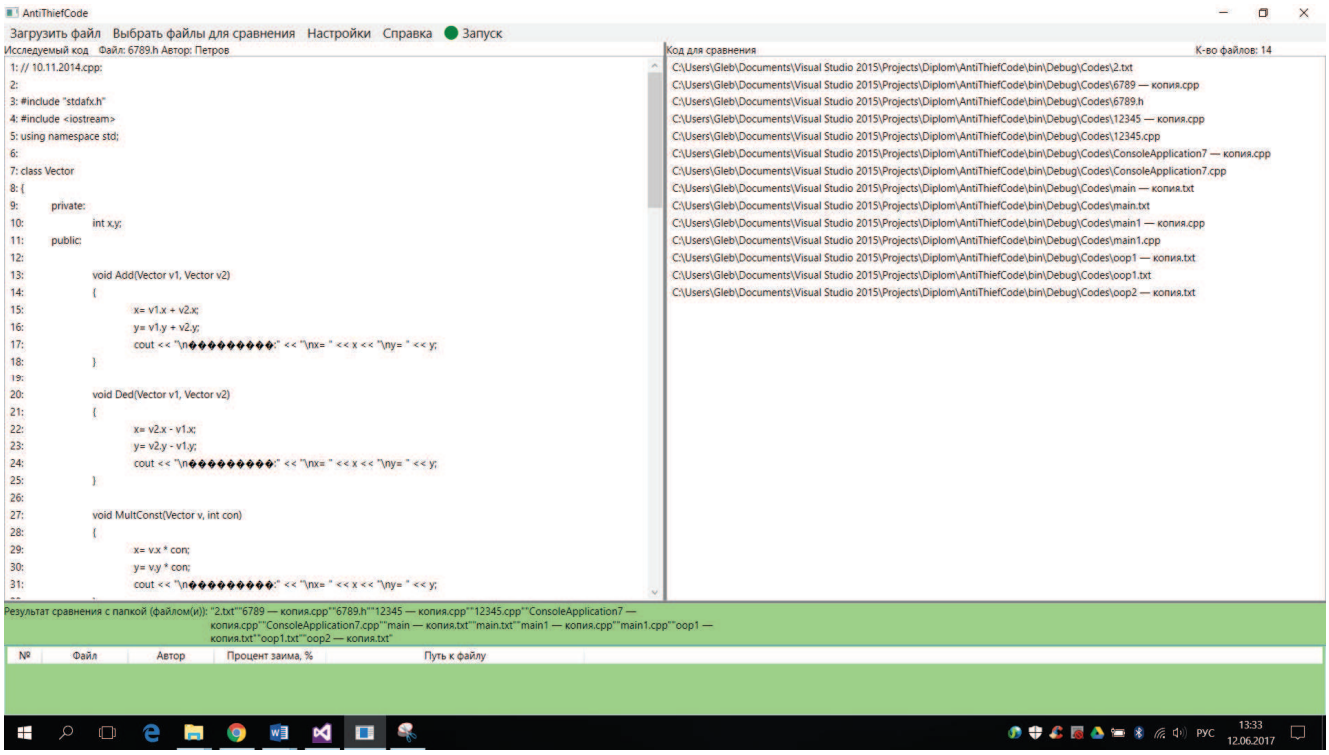


Рис. 13. Список добавленных файлов.

2.4.3. Сценарий настроек

1. Пользователь выбирает в панели меню вкладку «Настройки».
2. Панель раскрывается на еще два подменю (Рис. 14).

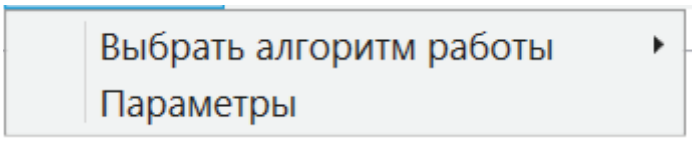


Рис. 14. Подменю настроек.

3. «Выбор алгоритмов работы» раскрывается на еще три подменю ().

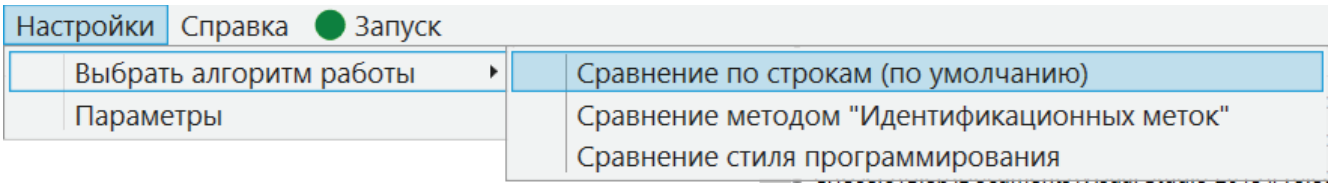


Рис. 15. Подменю выбора алгоритма.

4. Можно выбрать один из трёх представленных алгоритмов для сравнения. Для каждого из них открывается своё окно настроек.
5. «Параметры» выводят окно, где можно указать папку для сохранения файла конфигураций системы и хранилища обработанных файлов.

2.4.4. Справка

При выборе пункта меню «Справка», откроется окно справки, где рассказывается об данном проекте и даются инструкции для работы с ним.

2.4.5. Запуск

Запуск производится только если выполнены пункты 2.4.1, 2.4.2 и 2.4.3 (последний не обязателен, так как без него выбирается алгоритм работы и настройки по умолчанию).

После нажатия «Запуск» начинается процесс поиска плагиата в файлах, указанных для сравнения. По окончанию процесса в панели «Результат сравнения с папкой (файлом(и))» выводится список файлов, в которых был найден плагиат, авторы этих файлов, процент заимствования и полный путь к этим файлам. Также выводится общее количество сплагиаченных файлов и время выполнения процесса сравнения (Рис. 16).

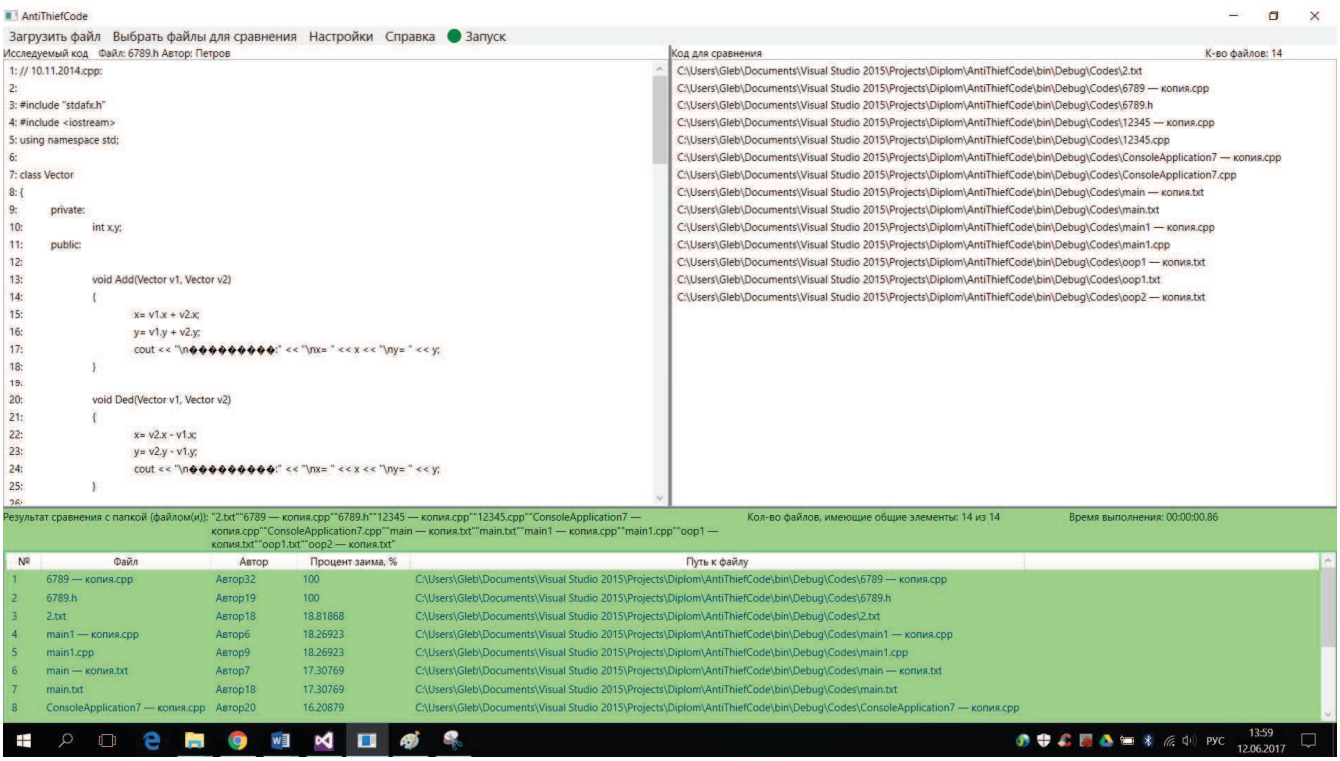


Рис. 16. Результат выполнения сравнений.

При нажатии на любой из элементов списка результата, содержимое в панели «Код для сравнения». Также выделятся красным цветом строчки похожих строк в обоих панелях с текстом исходного кода (Рис. 17).

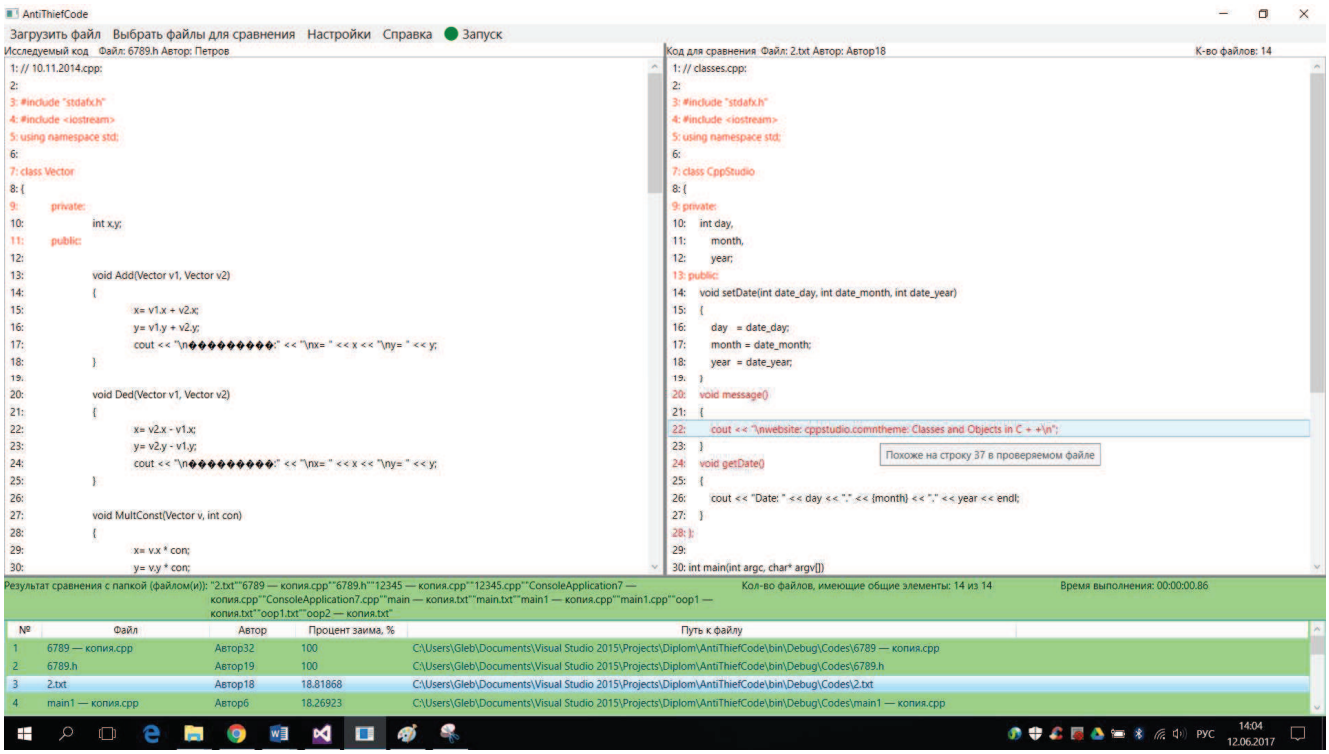


Рис. 17. Графическое изображение результата.

2.5. Требования к операционной системе

Программа может быть запущена на операционной системе Windows.

На персональных компьютерах системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 7 или выше.

2.6. Требование к аппаратной части

Система АИК разрабатывалась на ПК со стандартной комплектацией, среднего уровня. Так как работа происходит с текстовыми файлами, системе не требуется высокая производительность, достаточно будет ПК уровня офисной конфигурации.

3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Ключевой этап разработки приложения – проектирование. Для нашего программного продукта мы выделили следующие пункты:

- Проектирование библиотеки инструментов системы
- Программирование алгоритмов сравнения
- Разработка интерфейса

3.1. Проектирование библиотеки инструментов системы

Библиотека требуется для связывания двух проектов, консольного приложения и интерфейса и состоит из следующих компонентов:

- Инструменты для обработки текста – туда входят перечень регулярных выражений для поиска и замены фрагментов кода;
- Структуры представления фрагментов исходного кода – как рассматривалось в пункте 1.1 код требуется представить в одну из перечисленных моделей, но для объект каждой модели должен иметь определенный формат, для этого и требуется структуры представления фрагментов исходного кода;
- Алгоритмы сравнения исходных кодов – здесь прописаны сами алгоритмы сравнения;
- Работа с директориями и файлами – прописаны функции для обращения к файлам и папкам;
- Переопределение стандартных методов сравнения для работы со структурами представления фрагментов текста кода и расчет процента заимствования;
- Представление результата сравнения.

3.2. Программирование алгоритмов сравнения

В проекте используется три алгоритма сравнения, однако изучение данного вопроса продолжается и в дальнейшей их количество может возрасти.

Каждый из трех алгоритмов использует одну модель представления кода:

- Исходный код как есть
- Токенизация
- N-мерное пространство

3.2.1. Сравнение по строкам.

Для этого алгоритма используется модель «Исходный код как есть», но в параметризированном представление. То есть все переменные и функции меняются на свои значения, таким образом код предстает в общем виде. Далее происходит сравнение по строкам, одна строка со всеми остальными и выводится все общие строки. Затем количество символов в общих строках считается и делится на общее количество символов в исходном файле.

Трудоемкость данного алгоритма квадратная и он из всех трех считается самым быстрым, но точность оставляет желать лучшего.

3.2.2. Метод идентификационных меток.

Алгоритм был описан в 1.1.2.4. Но как выбрать идентификационные метки? Был выбран метод просеивания (winnowing) [27].

Алгоритм просеивания для построения меток. При поиске общей подстроки в файлах мы руководствуемся следующими условиями:

1. если длина совпадающей подстроки больше или равна гарантированной длине t, то совпадение будет обнаружено;
2. совпадения короче шумового порога k, игнорируются.

(Параметры t и k задают в зависимости от необходимой точности.)

Пункт 2 обеспечен выделением из текста k-граммов. Чем больше k, тем менее вероятно, что совпадения случайны. Но с ростом k падает устойчивость метода к перестановкам. В художественных текстах обычно за k принимают среднюю длину устойчивых выражений.

Чтобы удовлетворить пункту 1 необходимо (и достаточно), чтобы из каждых последовательно идущих (t - k + 1) хеш-значений хотя бы одно было выбрано в качестве метки.

Идея алгоритма: продвигаем окно размера w = (t - k + 1) вдоль последовательности hi... h, на каждом шаге окно перемещается на одну позицию вправо. Назначаем меткой минимальное hj в окне. Если в одном окне два элемента принимают минимальное значение, правый назначается меткой.

Проиллюстрируем этот процесс. Возьмем t = 5, k = 2:

A do run run run, a do run run

Простой текст

adorunrunrunadorunrun

Удалили все незначащие символы

adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador adoru dorun orunr runru unrun

Получим последовательность 5-граммов

77 74 42 17 98 50 17 98 8 88 67 39 77 74 42 17 98

Получим гипотетическую последовательность хэшей

(77, 74, 42, **17**) (74, 42, 17, 98) (42, 17, 98, 50) (**17**, 98, 50, 17) (98, 50, 17, 98) (50, 17, 98, **8**) (17, 98, 8, 88) (98, 8, 88, 67) (8, 88, 67, 39) (88, 67, **39**, 77) (67, 39, 77, 74) (39, 77, 74, **42**) (77, 74, 42, **17**) (74, 42, 17, 98)

Длина окна равна 4 хеш-значения

17 17 8 39 42 17

Метки, выбранные методом просеивания

[17, 3] [17, 6] [8, 8] [39, 11] [42, 14] [17, 15]

Метки в паре с позицией в первоначальном тексте.

Как видно, выбор метки определяется только содержимым окна, такой алгоритм называется локальным. Любой локальный алгоритм выбора меток корректен. Действительно, если в двух файлах есть достаточно большая общая подстрока, то будут и одинаковые окна, а значит, будут назначены одинаковые метки. По ним определим, что в файлах есть совпадения.

Показателем эффективности алгоритма может служить плотность d- доля хеш-значений, выбранных в качестве меток, среди всех хеш-значений документа. Можно показать [27], что при просеивании

$$d = 2/(w+l)$$

Трудоемкость (количество сравнений) зависит от заданного пользователем уровня точности.

3.2.3. Сравнение стилей программирования.

Данный алгоритм использует представление кода как точку n-мерного пространства. Но для определения стиля программирования одной точки мало, нужен набор программ одного автора и на основе этого набора строится общая точка n-мерного пространства, она будет уже числовыми характеристиками стиля.

В пункте 1.1.1 рассматривался способ определения плагиата используя представление n-мерное пространство, но там же было сказано, что данный метод не очень эффективен. Однако существует решение, если переопределить координаты точки не как характеристики кода, а как характеристики стиля (то есть отступы, регистр букв в названиях переменных, постановка открывающих\элементов блока кода), то можно получить хороший результат стиля программирования.

Есть несколько способов получения общей точки, например, можно просто взять среднюю точку, но как показала практика данный метод плохо определяет стиль. Другой способ — это отправить результирующие точки разных программ на обработку в нейронную сеть. Нейронные сети как раз принимают на вход множество параметров и хорошо обучаются.

Данный метод еще нуждается в изучении, но идея понятна.

3.3. Разработка интерфейса

В разработку интерфейса входят следующие задачи:

- Наглядное изображение результата;
- Удобное обращение к файлам и папкам;
- Устойчивость системы к ошибочным действиям пользователя.

Как говорилось в пункте 2.2.1, интерфейс разрабатывался на презентационной системе Windows Presentation Foundation. Все задачи были выполнены. Подробнее об этом в пункте 2.4.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

Программный комплекс АИК реализован на языке C# в среде разработки Visual Studio 2015 и состоит из двух частей:

- Библиотека инструментов обработки текстов и алгоритмов сравнения
- Интерфейс

Еще есть консольный проект для ускоренного тестирования программы, но о нем мы не будем говорить.

4.1. Библиотека инструментов обработки текстов

Карта кода библиотеки изображена на (Рис. 18).

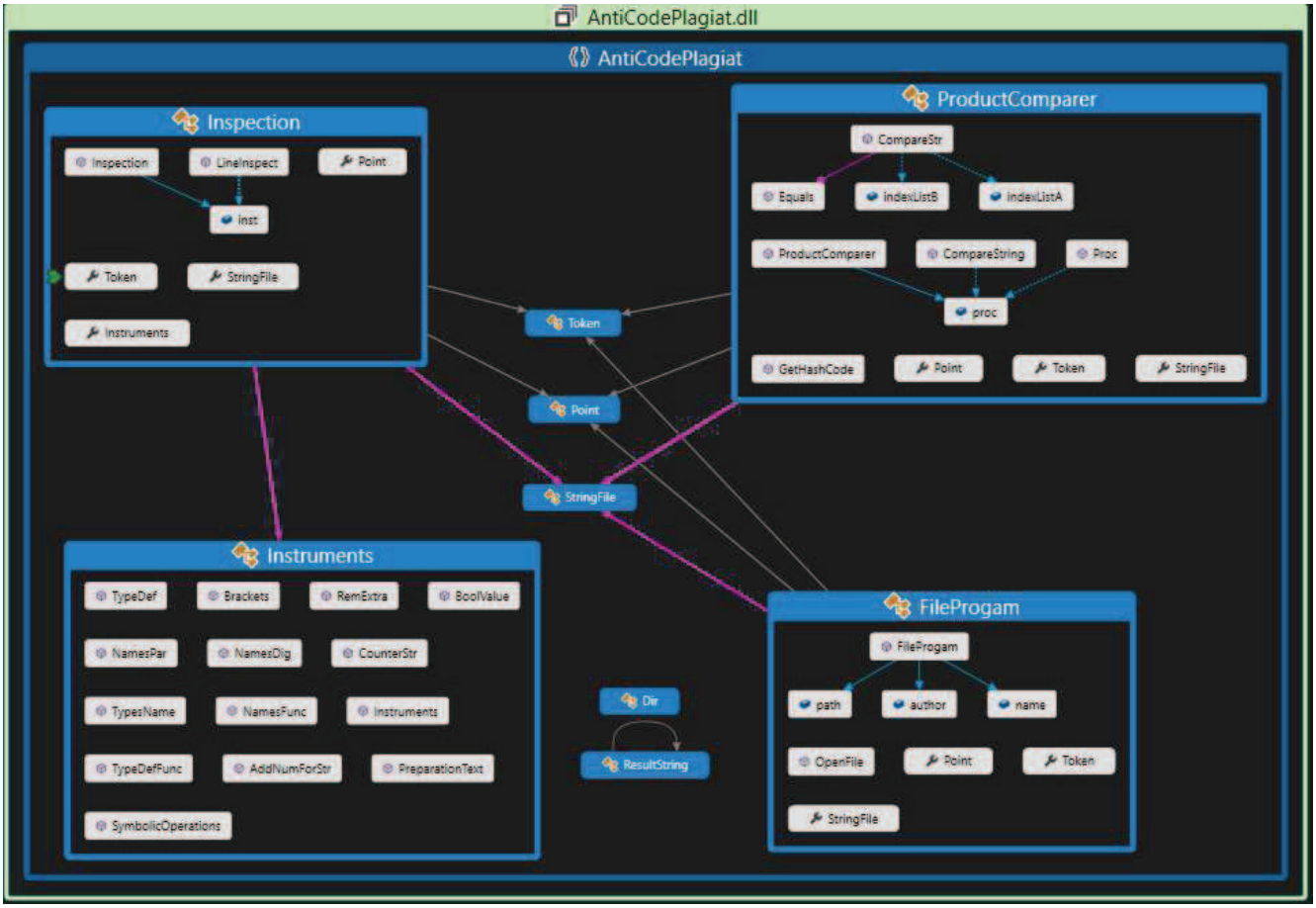


Рис. 18. Карта кода библиотеки инструментов

Как видно из рисунка, библиотека в данный момент состоит из девяти классов:

- Inspection – класс проверки, туда входят все алгоритмы сравнения
- Token – класс представления объекта токен, из таких строится строки для модели токенизации

- Point - класс представления объекта точки n-мерного пространства
- StringFile – переопределенный метод string, в него входят строки файлов и номера этих строк в файле, класс наследуется от System.Object
- ProductCompare – здесь переопределены стандартные методы сравнения объектов перечисленных выше
- Instrumets – класс, содержащий все методы обработки текста, для всех моделей, в основном это функции построенные на регулярных выражениях из библиотеки System.Text.RegularExpressions
- FileProgram – класс, для работы с файлами, с помощью него создаются структуры, содержащие имя файла, имя автора, расположение
- Dir – класс, для работы с папками, аналогичен FileProgram, но не имеет поле имени автора, с помощью него определяется директория, в которй буду производится сравнения
- ResultString – класс представления результата проверки, объекты данного класса заполняют панель «Результат сравнения» в интерфейсе

Пример алгоритма сравнения из класса ProductCompare:

```
public List<StringFile> CompareStr(ref List<StringFile> a, ref List<StringFile> b)
    {
        List<StringFile> duplicates = new List<StringFile>();
        indexListA = new List<int>();
        indexListB = new List<int>();
        if (a == (List<StringFile>)null)
            a = new List<StringFile>();
        if (b == (List<StringFile>)null)
            b = new List<StringFile>();
        foreach (var i in a)
        {
            foreach (var j in b)
            {
                if (Equals(i, j))
                {
                    indexListA.Add(i.string_numb);
                    indexListB.Add(j.string_numb);
                    duplicates.Add(j);
                    b.Remove(j);
                    break;
                }
            }
        }
        return duplicates;
        }
```

Листинг 1. Сравнение списка строк в файле.

Пример инструмента из класса Instrumets:

```
public string NamesFunc(ref string inputString) //на вход подается строка
        {
            if (inputString == (string)null) //проверяется, что строка не пустая
                inputString = "";
            string patternFuncFist =
@"(\w|\W)?(char|int|bool|string|void|long|float|short|unsigned\s(char|int|bool|string|void|lon
g|float|short))(\s+|\*|\&)(\s+|)(?<func>\w+)(\()"; //регулярное выражение для поиск имен
объявляемых функций в коде
            foreach (Match m in Regex.Matches(inputString, patternFuncFist)) //перечисление,
полученных значении
            {
                if (m.Groups["func"].Value != "main") //функция main не переименовывается
                {
                    string patternFunc = @"(\W)(" + m.Groups["func"].Value + @")(\W)";
//шаблон для поиска имен функция при вызове
                    string replacementFunc = "$1F$3"; // шаблон замены
                    inputString = Regex.Replace(inputString, patternFunc, replacementFunc); //
функция замены
                }
            }
            return inputString; // возврат обработанной строки
        }
```

Листинг 2. Замена имен функций

```
public string NamesPar(ref string inputString) //на вход подается строка
        {
            if (inputString == (string)null) //проверяется, что строка не пустая
inputString = "";
            string patternStringParFist =
@"(\w|\W)?(char|int|bool|string|void|long|float|short|unsigned\s(char|int|bool|string|void|lon
g|float|short))(\s+|\*|\&)(\s+|)(?<string_par>((\s+|)\w+(\s+|)(,|;|\)|\[))+)";
            string patternParFist = @"(?<par1>\w+)"; //регулярное выражение для поиск имен
объявляемых функций в коде

            string patternAssignment =
@"(\w|\W)?(char|int|bool|string|void|long|float|short|unsigned\s(char|int|bool|string|void|lon
g|float|short))(\s+|\*|\&)(\s+|)(?<assignment>\w+)(\s*\=\s)";

            foreach (Match m in Regex.Matches(inputString, patternStringParFist))
//перечисление, полученных значении

            {
                string patternPar = @"(\W)(" + m.Groups["string_par"].Value + @")(\W)";
                foreach (Match m1 in Regex.Matches(patternPar, patternParFist))
                {
                    string patternPar1 = @"(\W)(" + m1.Groups["par1"].Value + @")(\W)";
                    string replacement3 = "$1P$3";
                    inputString = Regex.Replace(inputString, patternPar1, replacement3); //
функция замены

                }
            }
            foreach (Match m in Regex.Matches(inputString, patternAssignment))
            {
                string patternPar = @"(\W)(" + m.Groups["assignment"].Value + @")(\W)";
                foreach (Match m1 in Regex.Matches(patternPar, patternParFist))
                {
                    string patternPar1 = @"(\W)(" + m1.Groups["par1"].Value + @")(\W)";
                    string replacement3 = "$1P$3";
```

```
                    inputString = Regex.Replace(inputString, patternPar1, replacement3); //
функция замены

                }
            }
            return inputString;
        }
```

Листинг 3. Замена имен переменных

4.2. Интерфейс

Интерфейс был разработан в WPF и имеет следующую карту кода:

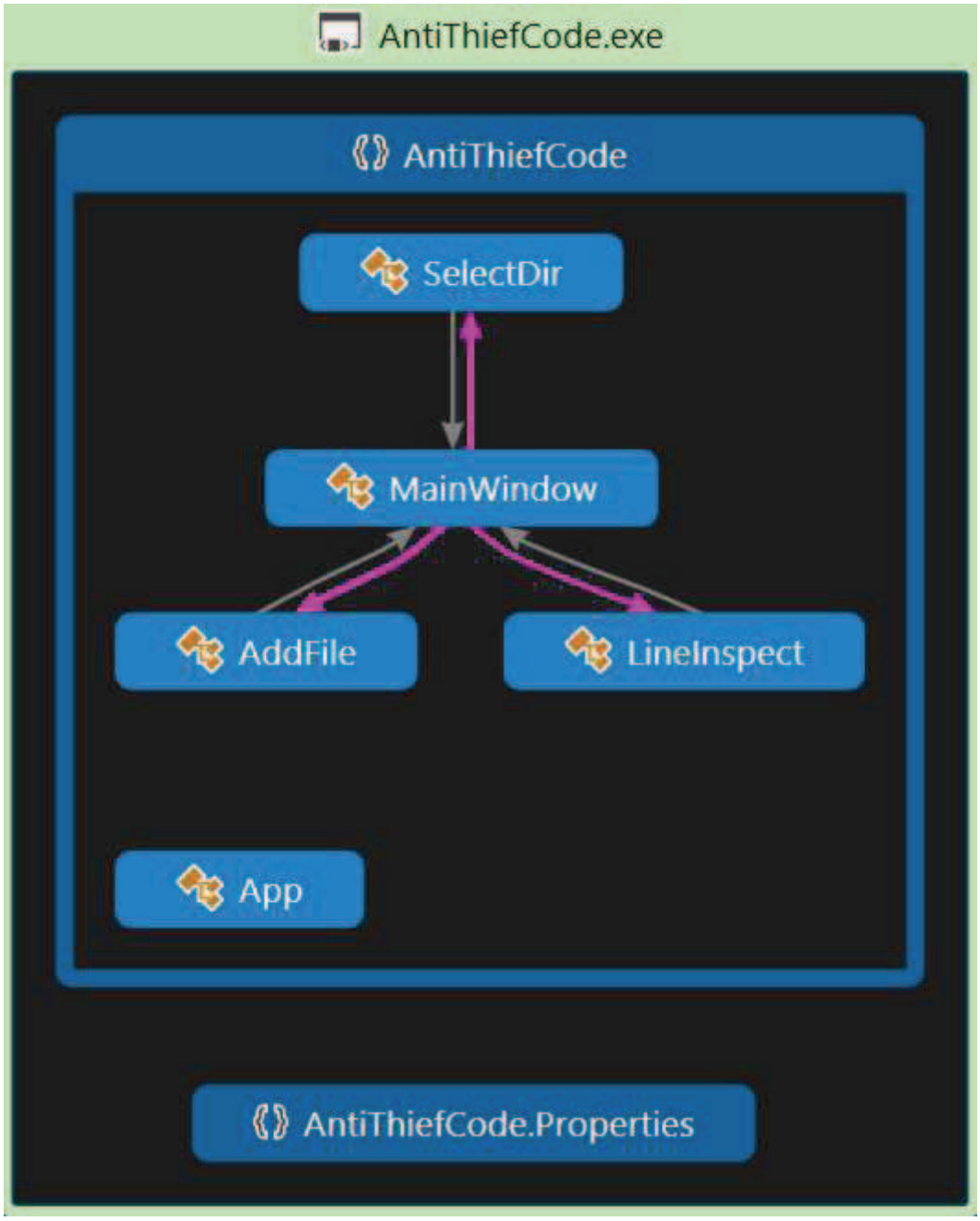


Рис. 19. Карта кода интерфейса

Как видно из рисунка (Рис. 19), в данный момент для интерфейса реализовано три класса:

MainWindow – главное окно программы (Рис. 20), здесь мы можем видеть три главных панели: «Исходный код», «Код для сравнения», «Результат сравнения с папкой (файлом(ами)):», что на них располагается можно узнать из пункта 2.4, там же расписано про меню. Рассмотрим подробно другие панели. Над левой панелью, справа от слов «Исходный код» присутствует поле, где отображается имя загруженного файла, имя автора. На правой панели аналогично возле слов «Код для сравнения», кроме этого в правой части правой панели указывается количество файлов, когда выбирается папка или файлы для сравнения. В панели результата, кроме его самого еще располагаются количество проверенных файлов и время выполнения.

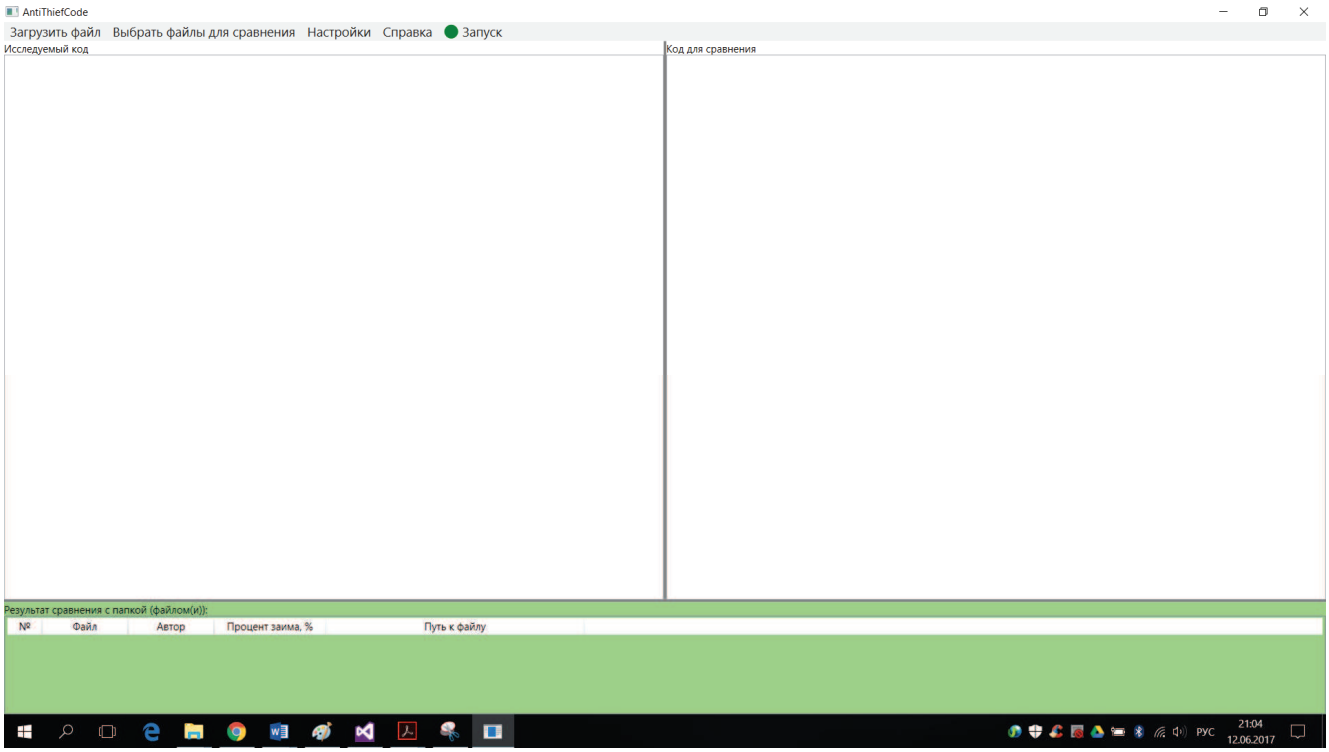


Рис. 20. Главное окно АИК.

Благодаря WPF окна интерфейса можно описывать в двух форматах, на языках C# и XAML. Пример кода главного окна на XAML:

```
<Window x:Class="AntiThiefCode.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AntiThiefCode"
        xmlns:col="clr-namespace:System.Collections;assembly=mscorlib"
```

```
        mc:Ignorable="d"
        WindowState="Maximized"
        Closing="MainWindow_Closing"
        Title="AntiThiefCode" Height="700" Width="1000">

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" MinWidth="300" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" MinWidth="300" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="0.2*"></RowDefinition>
        </Grid.RowDefinitions>
        <GridSplitter Grid.Column="1"  Grid.Row="1" ShowsPreview="False" Width="3"
        HorizontalAlignment="Center" VerticalAlignment="Stretch" Background="Gray" />
        <GridSplitter Grid.Row="2" Grid.ColumnSpan="3" Height="3"
        HorizontalAlignment="Stretch" VerticalAlignment="Center" Background="Gray" />
```

Листинг 4. Фрагмент кода главного окна на XAML

На листинге 4 приведен небольшой отрывок кода главного окна АИК, здесь указаны габариты и способ открытия. Окно MainWindow по умолчанию открывается в полный экран, но способно изменять свой размер, без вреда для содержимого окна.

AddFile – окно загрузки файла (Рис. 10), описание окна приведено в пункте 2.4.1. Пример кода окна загрузки файла на XAML:

```
<Window x:Class="AntiThiefCode.AddFile"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AntiThiefCode"
        mc:Ignorable="d"
        WindowStyle="ToolWindow"
        ResizeMode="CanMinimize"
        Title="Добавление файла" Height="130" Width="640">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="0.5*"></RowDefinition>
            <RowDefinition Height="0.6*"></RowDefinition>
            <RowDefinition Height="0.1*"></RowDefinition>
            <RowDefinition Height="0.6*"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="0.3*"/>
        </Grid.ColumnDefinitions>
        <TextBox x:Name="tb_Path" Width="480" Grid.ColumnSpan="2" Grid.Row="1" Margin="4 0"
HorizontalAlignment="Left"></TextBox>
        <Button x:Name="btn_AddFile" Content="Добавить файл" Grid.Column="1" Grid.Row="1"
Click="btn_AddFile_Click" Margin="4 0"></Button>
```

```
        <TextBox x:Name="tb_Author" Width="480" Grid.ColumnSpan="2" Grid.Row="3" Margin="4 0"
HorizontalAlignment="Left"></TextBox>
        <TextBlock Grid.Column="1" Grid.Row="3" HorizontalAlignment="Center" Margin="4
0">Введите имя автора</TextBlock>
        <Button x:Name="btn_OK" Content="OK" Grid.Column="0" Grid.Row="4"
HorizontalAlignment="Center" Margin="0 0 0 10" VerticalAlignment="Bottom" Width="75"
Click="btn_OK_Click"/>
        <Button x:Name="btn_Cancel" Content="Отмена" Grid.Column="1" Grid.Row="4"
HorizontalAlignment="Center"  Margin="0 0 0 10" VerticalAlignment="Bottom" Width="75"
Click="btn_Cancel_Click"/>

    </Grid>
    </Window>
```

Листинг 5. Код окна загрузки файла на XAML

SelectDir – окно выбора папки или файлов для сравнения с ними (Рис. 12), описание окна приведено в пункте 2.4.2. Пример кода окна загрузки файла на XAML:

```
<Window x:Class="AntiThiefCode.SelectDir"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AntiThiefCode"
        mc:Ignorable="d"
        WindowStyle="ToolWindow"
        ResizeMode="CanMinimize"
        Title="Выбрать файлы для сравнения" Height="130" Width="640">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="0.5*"></RowDefinition>
            <RowDefinition Height="0.5*"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="0.3*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.ColumnSpan="2" Grid.Row="0" Orientation="Horizontal">
            <RadioButton x:Name="rb_Dir" GroupName="FilesOrDir" Content="Выбрать паку"
Margin="10 4" IsChecked="True" ToolTip="Выберите любой файл и автоматически выбирется каталог,
где лежит этот файл"></RadioButton>
            <RadioButton x:Name="rb_Files" GroupName="FilesOrDir" Content="Выбрать файл(ы)"
Margin="10 4"></RadioButton>
        </StackPanel>
        <TextBox x:Name="tb_Path" Width="480" Grid.ColumnSpan="2" Grid.Row="1" Margin="4 0"
HorizontalAlignment="Left" ToolTip="Здесь задается путь к папке, в которой храни(я)тся
выбранный(е) файл(ы)"></TextBox>
        <Button x:Name="btn_SelectDir" Content="Выбрать файл(ы)" Grid.Column="1" Grid.Row="1"
Click="btn_SelectDir_Click" Margin="4 0"></Button>
        <Button x:Name="btn_OK" Content="OK" Grid.Column="0" Grid.Row="3"
HorizontalAlignment="Center" Margin="0 0 0 10" VerticalAlignment="Bottom" Width="75"
Click="btn_OK_Click"/>
        <Button x:Name="btn_Cancel" Content="Отмена" Grid.Column="1" Grid.Row="3"
HorizontalAlignment="Center"  Margin="0 0 0 10" VerticalAlignment="Bottom" Width="75"
Click="btn_Cancel_Click"/>

    </Grid>
    </Window>
```

Листинг 6. Код окна выбора файлов для сравнения на XAML

LineInspect – представляет собой окно настроек алгоритма сравнения по строкам (Рис. 21).

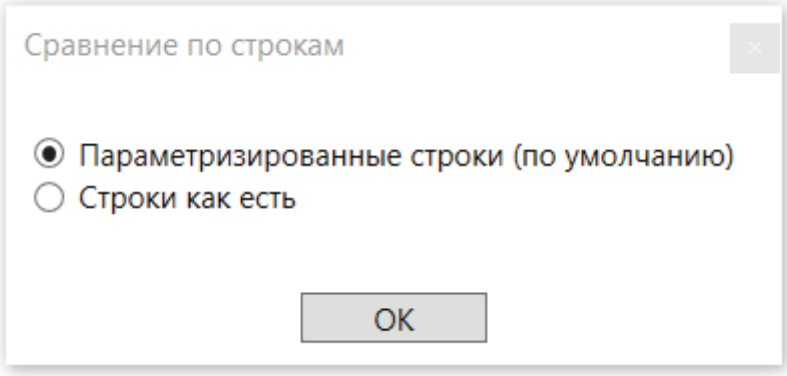


Рис. 21. Окно настроек алгоритма сравнения по строкам

В данном окне можно выбрать два режима работы алгоритма сравнения.

Пример кода данного окна:

```
<Window x:Class="AntiThiefCode.LineInspect"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AntiThiefCode"
        mc:Ignorable="d"
        WindowStyle="ToolWindow"
        ResizeMode="CanMinimize"
        Title="Сравнение по строкам" Height="150" Width="324">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="0.5*"></RowDefinition>
        </Grid.RowDefinitions>
        <StackPanel Grid.Row="0" VerticalAlignment="Center">
            <RadioButton x:Name="rb_Param" Margin="10 1" GroupName="Mode"
Content="Параметризированные строки (по умолчанию)" IsChecked="True"></RadioButton>
            <RadioButton x:Name="rb_Simple" Margin="10 1" GroupName="Mode" Content="Строки как
есть"></RadioButton>
        </StackPanel>
        <Button x:Name="btm_OK" Grid.Row="1" HorizontalAlignment="Center"
VerticalAlignment="Center" Content="OK" Width="75" Click="btm_OK_Click"/>

    </Grid>
    </Window>
```

Листинг 7. Код окна настроек алгоритма сравнения по строкам

5. ТЕСТИРОВАНИЕ

5.1.Функциональное тестирование

Функциональное тестирование – это тестирование программное обеспечение (ПО) в целях проверки реализуемости функциональных требований, то есть способность ПО в определенных условиях решать задачи, нужные пользователям, а также оно определяет, что именно делает ПО, какие задачи оно решает [37].

Тест №1. Сравнение двух файлов исходного кода

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и хочет сравнить два файла на заимствования.

Ожидание: пользователь без затруднений должен добавить исходный файл, выбрать файл для сравнения и нажать «Запуск», результат отобразится в нижней панели.

Полученный результат: пользователь без затруднений добавляет исходный файл, выбирает файл для сравнения и нажимает «Запуск», результат отображается в нижней панели.

Тест №2. Сравнение одного файла с несколькими файлами исходного кода

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и хочет сравнить один файл с несколькими файлами исходного кода.

Ожидание: пользователь без затруднений должен добавить исходный файл, выбрать несколько файлов для сравнения и нажать «Запуск», результат отобразится в нижней панели.

Полученный результат: пользователь без затруднений добавляет исходный файл, выбирает несколько файлов для сравнения и нажимает «Запуск», результат отображается в нижней панели.

Тест №3. Сравнение одного файла с папкой файлов исходного кода

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и хочет сравнить один файл со всеми файлами конкретной папки исходных кодов.

Ожидание: пользователь без затруднений должен добавить исходный файл, выбрать папку для сравнения и нажать «Запуск», результат отобразится в нижней панели.

Полученный результат: пользователь без затруднений добавляет исходный файл, выбирает папку для сравнения и нажимает «Запуск», результат отображается в нижней панели.

Тест №4. Сравнение одного файла с папкой файлов, в которую входит сто файлов исходного кода

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и хочет сравнить один файл со всеми файлами конкретной папки исходных кодов, в которой находится сто файлов.

Ожидание: пользователь без затруднений должен добавить исходный файл, выбрать папку для сравнения и нажать «Запуск», результат отобразится в нижней панели, на выполнения алгоритма уйдет около 40 секунд.

Полученный результат: пользователь без затруднений добавляет исходный файл, выбирает папку для сравнения и нажимает «Запуск», результат отображается в нижней панели, время выполнения 58 секунд.

5.2. Тестирование интерфейса

Тестирование интерфейса необходимо проводить для того, чтобы избежать:

- Потери или искажения элементов пользовательского интерфейса
- Ошибок в интерфейсе
- Некорректный результат интерфейса.

Тест №1. Запуск проверки без выбора исходного файла и файлов для сравнения

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и нажимает кнопку «Запуск», не загрузив предварительно исходный файл и не выбрав ни одного файла для сравнения.

Ожидание: пользователь нажмет на кнопку «Запуск», система выведет ошибку.

Полученный результат: отображен на (Рис. 22).

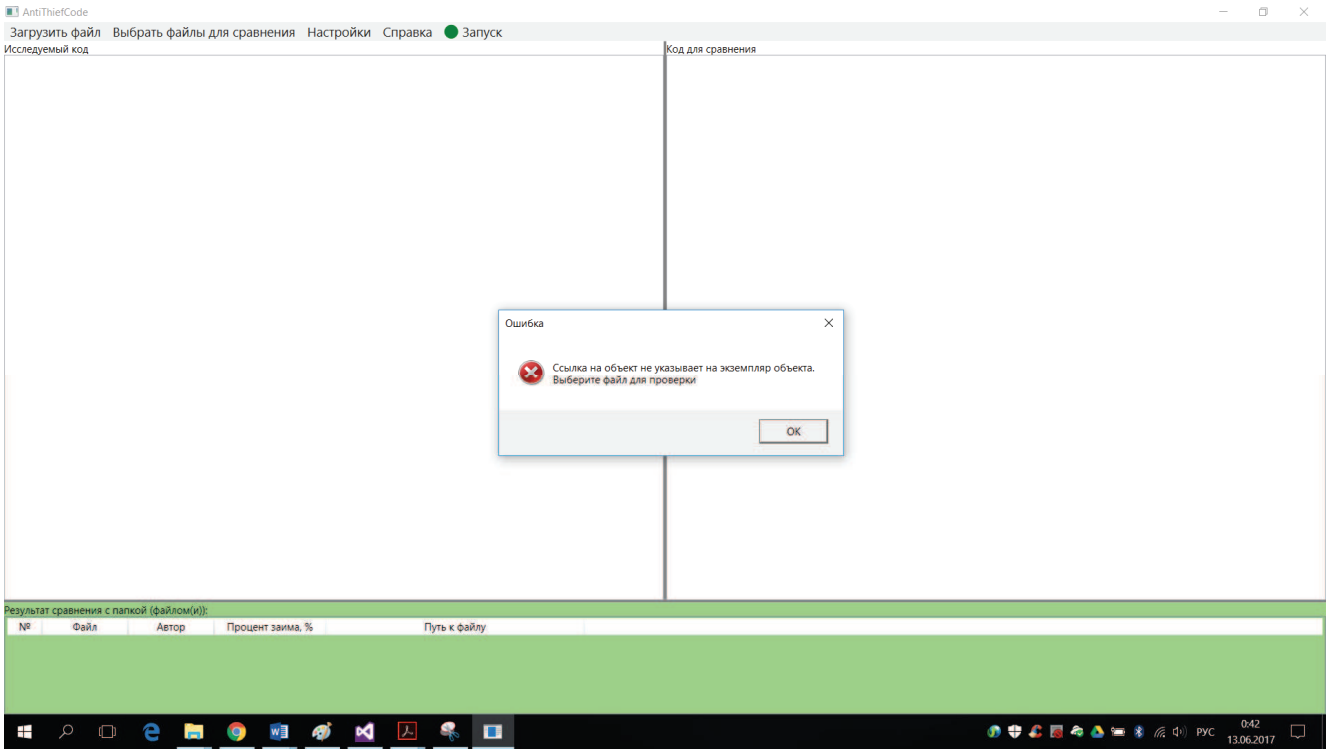


Рис. 22. Система выдает ошибку

Тест №2. Добавления файла исходного кода без указывания автора

Входные данные: пользователь открыл систему Антиплагиат исходных кодов и пытается добавить новый файл исходного кода без указания автора программы.

Ожидание: пользователь нажмет на кнопку «ОК», система выведет ошибку.

Полученный результат: отображен на (Рис. 23).

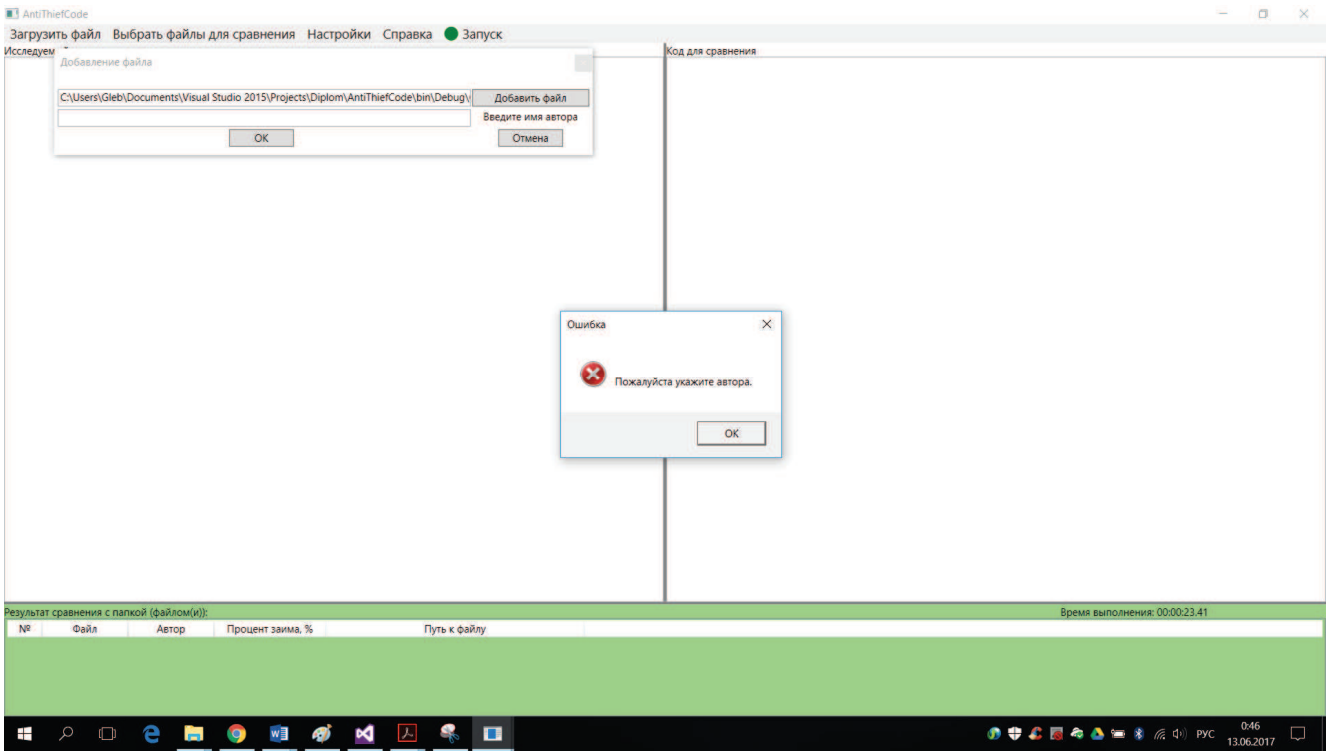


Рис. 23. Система выдает ошибку.

ЗАКЛЮЧЕНИЕ

Целью работы являлась разработка системы Антиплагиат исходных кодов.

Для достижения данной цели были решены следующие задачи:

- Изучены технологии программирования алгоритмов поиска плагиата исходных кодов
- Произведен обзор существующих решений
- Выбран язык программирования C#
- Выбрана платформ реализации Windows
- Выбрана среда разработка Visual Studio
- Определены требования к операционной системе
- Спроектирована система Антиплагиат исходных кодов
- Разработаны алгоритмы сравнения
- Разработан интерфейс

Поставленные задачи успешно выполнены, цель достигнута. Простой интерфейс программы не вызовет трудности для начинающего пользователя. В дальнейшем планируется развитие системы в сторону добавление новых алгоритмов поиска плагиата, повышения точности работы.

В ближайшем будущем планируется реализовать еще два алгоритма сравнения, указанных в пункте [3.2].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ожегов, С. И. Словарь русского языка. – Москва: «РУССКИЙ ЯЗЫК», 1986. – 797 стр.
2. Синельников, С.М. Энциклопедия предпринимателя. – СПб., 1994.
3. Сухарева, А.Я., Крутских, В.Е. Большой юридический словарь. – М., 2002.
4. Интернет-сервис Антиплагиат.Ру [Электронный ресурс]. – Режим доступа – URL: http://www.antiplagiat.ru/index.aspx.
5. Антиплагиат.ВУЗ [Электронный ресурс]. – Режим доступа – URL: http://susu.antiplagiat.ru/index.aspx.
6. J. L .Donaldson, A. Lancaster and P. H. Sposato, A plagiarism detection system, ACM SIGSCI Bulletin 13(1), February 1981 – pp(21-25).
7. West A. Copying with plagiarism in Computer Science teaching laboratories, Computers in Teaching Conference. – Dublin, 1995.
8. Prechelt. L., Malpohl, G., Philippsen M., JPlag: Finding Plagiarisms among a Set of Programs, Technical Report 2000-1, Fakultat fur Informatik, Universitat Karlsruhe, 2000.
9. Schleimer S., Wilkerson D. S., Aiken A. Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data. – San Diego, California, June 09-12, 2003.
10. Whale G. Identification of program Similarity in Large Populations, The Computer Journal, Vol.33. – Number 2, 1990.
11. Fotel C., Langer L. A Plagiarism Detection Tool. – May 19, 2004.
12. M. Joy and Michael Luck, Plagiarism in Programming Assignments, IEEE Transactions on education. AMI. 42, NO. 2, May 1999.
13. Chen X., Francia B., Li M., Mckonnon B., Seker A. Shared information and program plagiarism detection. – University of California, Santa Barbara, December 13, 2003.
14. Udi Manber and Brenda S. Baker, Deducing similarities in Java sources from bytecode, 1998 USENIX Technical Conference. – New Orleans, June 1998.

15. David Gitchell and Nicholas Tran, Sim: A utility for detecting simile,rity in computer programs, Wichita State Univercity.
16. Dick Grune, Matty Huntjens, Het detecteren van kopieen bij informatica-practica, – Nov 1989, pp. 864-867.
17. Culwin F. and Naylor J., Pragmatic Anti-Plagiarism, Proceedings Third Conference on the Teaching of Computing. – DCU Dublin IE 1995.
18. Michael J. Wise, Detection of similarities in student programs: YAP'ing may be prefarable to Plague'ing, SIDSCI Technical Symposium. – Kansas City, USA, March 5-6, 1992 – pp 268,271.
19. Michael J. Wise, String similarity via Greedy Tiling and Running Karp-Rabin Matching, an unpublished paper. – December 1993.
20. Холстед M. H. Halstead, Elements of software science, – North Holland, New York, 1977.
21. Мак Кейб T. J. McCabe, A complexity measure, IEEE Transactions on Software Engineering, SE-2 (4). – December 1976. – pp 308-320.
22. Verco, K. and Wise, M. — (1996) — Software for detecting suspected plagiarism: comparing structure and attribute counting systems, In Proc. First Australian Conf. on Computer Science Education, Sydney Australia, July 3-5, — pp. 86-95. ACM Press, New York, USA.
23. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий = Compilers: Principles, Techniques, and Tools. — 2-е изд. — М.: Вильямс, 2008. — ISBN 978-5-8459-1349-4.
24. Фридл, Дж. Регулярные выражения = Mastering Regular Expressions. — СПб.: «Питер», 2001.
25. N. Heintze, Scalable document fingerprinting. In 1996 USENIX Workshop on Electronic Commerce, 1996.
26. U. Manber, Finding similar files in a large file system. In Proceedings of the USENIX Winter 1994 Technical Conference, pages 1-10, San Francisco, CA, USA, 1994.

27. A. Aiken, S. Schleimer, D. Wikerson, Winnowing: local algorithms for document fingerprinting. In Proc 2003 ACMSIGMOD Int. Conf. on Management of Data, San Diego, CA, June 9-12, pp. 76-85. ACM Press, New York, USA, 2003.
28. Prechelt. L., Malpohl, G., Philippsen M., JPlag: Finding Plagiarisms among a Set of Programs, Technical Report 2000-1, Fakultat fur Informatik, Universitat Karlsruhe, 2000.
29. Saul Schleimer , Daniel S. Wilkerson , Alex Aiken, Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 09-12, 2003, San Diego, California.
30. Бьёрн Страуструп. Язык программирования C++ = The C++ Programming Language / Пер. с англ. — 3-е изд. — СПб.; М.: Невский диалект — Бином, 1999. — 991 с. — 3000 экз. — ISBN 5-7940-0031-7 (Невский диалект), ISBN 5-7989-0127-0 (Бином), ISBN 0-201-88954-4 (англ.).
31. Maureen Williams Zimmerman. Microsoft Visual Basic 6.0 Reference Library. — Redmond, WA, USA: Microsoft Press, 1998. — 3344 с. — ISBN 1-57231-864-3.
32. Герберт Шилдт. Java 8. Полное руководство, 9-е издание = Java 8. The Complete Reference, 9th Edition. — М.: «Вильямс», 2015. — 1376 с. — ISBN 978-5-8459-1918-2.
33. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: «Диалектика», 2013. — 1440 с. — ISBN 978-5-8459-1850-5.
34. URL: https://www.microsoft.com/ru-ru/windows/.
35. URL: https://3dnews.ru/tags/%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0/page-6.html.
36. Windows Presentation Foundation [Электронный ресурс]. – Режим доступа – URL:https://msdn.microsoft.com/en-us/library/ms754130(v=vs.100).aspx

37. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Tester and Agile Teams. /Л. Криспин. – М.: «Вильямс», 2010. – 464 с. – 1000 экз.