УДК 004.43+004.9

А.А. Киричек[1], А.А. Амонс[1], Г.Г. Киричек[2],
[1]Национальный технический университет Украины "КПИ", г. Киев
[2]Запорожский национальный технический университет, г. Запорожье

ФИЛЬТРАЦИЯ ШАБЛОНОВ ПРОГРАММНОГО КОДА В СТУДЕНЧЕСКИХ ПРОЕКТАХ

Аннотация

***Киричек А.А., Амонс А.А., Киричек Г.Г.,* Фильтрация шаблонов программного кода в студенческих проектах.** *Разработан алгоритм фильтрации, как часть системы определения плагиата в программном коде. Исследование посвящено решению задачи отсечки файлов шаблона и фрагментов программного кода проекта до применения основных алгоритмов оценки подобия.*

Ключевые слова: *плагиат, программирование, токенизация, синтаксический анализ, фильтрация*

Постановка задачи. Задачей системы оценки идентичности программного кода, является автоматическое обнаружение (по заданным критериям) того, была ли использована в программе чужая идея. На практике определенным образом задаются функция близости и порог, по которым можно определить вероятность заимствования части кода [1]. В соответствии с [2], принято выделять следующие подходы к оценке близости программ: атрибутно-подсчетный, структурный и комбинированный.

Все текущие алгоритмы ориентированы на поиск плагиата в коде отдельных файлов. Зачастую проекты студентов состоят из множества файлов программного кода, иногда использующих стандартные шаблоны [3].

Актуальность данного исследования в необходимости уточнения текущих алгоритмов определения повторного использования программного кода, за счет применения к разработанному проекту алгоритма фильтрации, запускаемого на этапе импортирования в систему. Благодаря реализации этого алгоритма появится инструмент, позволяющий импортировать в систему проект целиком и получать высокую точность оценки наличия в нем плагиата.

В работе рассматривается решение поставленной задачи для проектов, написанных на языке программирования C# [4]. Большая часть кода программ на C# имеют идентичную реализацию так как они созданы на шаблонах проектов предлагаемых средой разработки. Поэтому для выделения уникальной составляющей, выполненного студентом задания, следует применить отсеивание. Удаление из проекта файлов программного кода шаблона позволит оценить изменения, внесенные конкретным разработчиком.

Реализация предложенного алгоритма фильтрации состоит из двух

частей. Первая исключает определенные файлы из проекта, которые являются частью предопределенных шаблонов, на основе которых он был создан. Вторая либо полностью исключает, либо существенно уменьшает вклад в общий процент подобия участков кода, соответствующих определенным шаблонам среды Visual Studio, либо реализации паттернов проектирования [1]. Комплексное применение данных методов позволит повысить точность последующей оценки подобия программного кода. Алгоритм обработки проекта, перед проведением его анализа, представлен на рисунке 1.

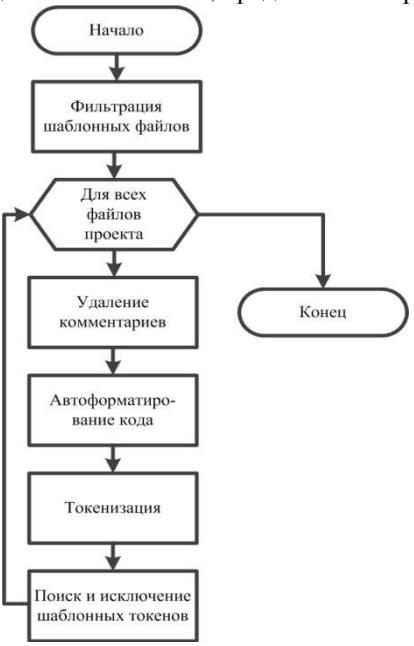

Рисунок 1 - Алгоритм обработки проекта перед анализом на плагиат

Его суть в проведении фильтрации шаблонных файлов проекта. Основные этапы для всех файлов кода, включают в себя выполнение последовательности операций, состоящих из: удаления комментариев, автоформатирования, токенизации, а также поиска и исключения шаблонных токенов.

Реализация алгоритма. В процессе получения информации о заимствовании определенных участков программного кода в разработанных проектах, имеется возможность произвести их оценку и представить отчет об использовании идентичных файлов ресурсов, либо отдельно взятых модулей приложения. Это даст возможность преподавателю более точно оценить уникальный вклад студента в решении, поставленной перед ним задачи[5].

Для исключения файлов шаблонных проектов, используется метод поиска в каталоге проекта предопределенных файлов, входящих в состав каждого шаблона. Предустановленные шаблоны можно дополнять пользовательскими, которые становятся доступны при создании нового проекта.

Для примера можем рассмотреть шаблон WPF MVVM project template. Проект включает в себя список файлов, которые должны быть исключены из проверки, но только при соблюдении условия, что в них не вносятся изменения в процессе разработки самого программного проекта. Данный метод можно применять к любому типу шаблонов. При необходимости можно проанализировав информацию о вычитании определенных шаблонных файлов из проекта предоставить отчет о проценте использования стандартных шаблонов. После того, как не подлежащие изменению файлы исключены из проекта, необходимо проанализировать все оставшиеся файлы проекта на наличие фрагментов кода, состоящих из стандартных шаблонов.

Первым этапом подготовки файла к исследованию является проведение его очистки от комментариев. Наличие подобия данных участков не должно оказывать влияние на общее оценивание разработанного проекта.

В данном случае исключению подлежат блочные и строковые комментарии. Ниже приведен код программы, выполняющий очистку комментариев с учетом исключений:

```
    public string DeleteComments(string input)
        {
            var blockComments = @"/\*(.*?)\*/";
            var lineComments = @"//(.*?)\r?\n";
            var strings = @"""((\\[^\n]|[^""\n])*)""";
            var verbatimStrings = @"@(""[^""]*"")+";
            return Regex.Replace(input, blockComments +
"|" + lineComments + "|" + strings + "|" +
verbatimStrings,
                me =>
                {
                    if (me.Value.StartsWith("/*") ||
me.Value.StartsWith("//")) return
me.Value.StartsWith("//") ? Environment.NewLine : "";
                    // Keep the literal strings
                    return me.Value;
                },
                RegexOptions.Singleline);  }
```

Обязательным является выполнение этапов автоформатирования кода. В результате ее применения исключаются ошибки при проведении процесса токенизации и, соответственно, уменьшается сложность реализации самого процесса. Выполнить процесс автоформатирвания можно, воспользовавшись средствами Microsoft Visual Studio. Для этого используется библиотека EnvDTE.dll. Это COM библиотека, она содержит объекты и элементы для

автоматизации ядра Visual Studio. Ниже приведен код, позволяющий выполнить команды автоформатирования для нужного файла:

```
    addedItem =
project.ProjectItems.AddFromTemplate(file.FullName,
fileName);
    addedItem.Open(Constants.vsViewKindCode);
    addedItem.Document.Activate();
    addedItem.Document.DTE.ExecuteCommand("Edit.FormatDoc
ument");
    addedItem.SaveAs(file.FullName);
```

Далее необходимо удалить из файлов шаблонные участки кода. Этого можно достичь, исключив токенизированные представления этих фрагментов.

Рассмотрим шаблон проекта Windows Presentation Foundation:

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;
    using System.Windows;
    using System.Windows.Controls;
    namespace WpfApplication2
    {
        /// <summary>
        /// Interaction logic for MainWindow.xaml
        /// </summary>
        public partial class MainWindow : Window
        {
            public MainWindow()
            {
                InitializeComponent();
            }    }}
```

Далее приведем токенизированное представление этого фрагмента программы. Допустим, что определению области имен соответствует токен a, модификаторам public и partial токены b и c, а ключевому слову class токен e, тогда вложенный токен будет иметь вид: a{bcd{b}}}, который при указании k-грамма будет приравниваться одному символу. Например если k=3, то он будет иметь вид (рис. 2), где a{bcd{b}} –первый символ , k-второй, m-третий.

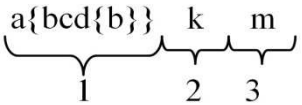

Рисунок 2 - Вложенный токен

Этот фрагмент кода повторяется для многих файлов в проекте при условии, что они используют паттерн проектирования Model-View-ViewModel.

Данный паттерн проектирования используется для разделения модели и её представления. Это необходимо для осуществления их независимости друг от друга в процессе внесения в них изменений. Например, когда программист задает или корректирует логику работы с данными, а дизайнер, соответственно, вносит изменения в работу с пользовательским интерфейсом.

Таким образом, эти повторяемые фрагменты кода могут повысить процент подобия двух абсолютно разных программных проектов. Из этого следует, что их необходимо тоже исключить из процесса проведения проверки на наличие плагиата в программном коде.

Если же подобные файлы содержат логику обработки событий, то они должны принимать участие в процессе проведения оценки уровня подобия.

Пример сигнатуры метода приведен ниже:

```
    private void btnSave_Click(object sender,
RoutedEventArgs e)
    {
    }
```

При этом содержание файлов с расширением ".xaml.cs" должно быть проанализировано на наличие единственного составного токена.

Автоматически генерируемые файлы с расширением ".Designer.cs" также должны быть исключены из процесса проведения анализа программного кода, если их содержимое соответствует уже предопределенному токену.

Рассмотрим применение алгоритма для данного фрагмента программы:

```
// ------------------------------------------------------
// <copyright file="MainWindow.xaml.cs" company="NDIIP">
//   Copyright © NDIIP 2013.
// </copyright>
// <author>Kirichek A.A.</author>
// ------------------------------------------------------
using System.Windows;
using BookReader.ViewModel;
using BookReader.Model.CommonLogic;
namespace BookReader.View.Views
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml.
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            MainViewModel vm = new MainViewModel();
            Serializer ser = new Serializer();
```

```
            vm.CommonBookFormatList =
ser.DeSerializeBooks("BooksFromLastSession.txt");
            foreach (CommonBookFormat item in
vm.CommonBookFormatList)
            {
                vm.OpenCommonFormat(item);
            }
            DataContext = vm;
        }
        private void Tombstoning()
        {
            Serializer ser = new Serializer();

ser.SerializeBooks("BooksFromLastSession.txt",
(DataContext as MainViewModel).CommonBookFormatList);
            (DataContext as
MainViewModel).SerializeRecentBooks();
        }
        /// <summary>
        /// Handling event of closing application by
menu option.
        /// </summary>
        private void Window_Closing(object sender,
RoutedEventArgs e)
        {
            Tombstoning();
            App.Current.Shutdown();
        }
    }
}
```

В результате применения алгоритма получен участок кода, который содержит лишь внесенные по сравнению с шаблоном изменения. Результирующее отображение кода сохраняется в виде последовательности токенов. Приведенный ниже фрагмент программы соответствует полученной последовательности:

```
    using BookReader.ViewModel;
    using BookReader.Model.CommonLogic;
    MainViewModel vm = new MainViewModel();
    Serializer ser = new Serializer();
    vm.CommonBookFormatList =
ser.DeSerializeBooks("BooksFromLastSession.txt");
    foreach (CommonBookFormat item in
vm.CommonBookFormatList)
    {
```

```
            vm.OpenCommonFormat(item);
        }
        DataContext = vm;
        private void Tombstoning()
        {
            Serializer ser = new Serializer();
            ser.SerializeBooks("BooksFromLastSession.txt",
(DataContext as MainViewModel).CommonBookFormatList);
            (DataContext as MainViewModel)
.SerializeRecentBooks();
        }
        Tombstoning();
        App.Current.Shutdown();
```

Выводы. В результате проведенного исследования предложен алгоритм фильтрации как часть системы определения плагиата в программном коде. Решена задача отсечки файлов шаблона и фрагментов программного кода, в разработанных программных проектах, до применения основных алгоритмов оценки подобия. Полученный алгоритм обеспечивает исключение из проекта неизменяемых фрагментов кода, являющихся шаблонами, в результате чего увеличена точность определения наличия плагиата в программном коде. Также при подготовке файлов проекта к последующему анализу на плагиат в данный алгоритм включены, пошагово, процессы автоформатирования и удаления из кода комментариев. Данный алгоритм фильтрации подлежит реализации как часть системы определения плагиата в программном коде, разработанных студентами проектов.

Список литературы

1. Киричек Г.Г. Модель системи оцінки ідентичності програмного коду [Текст] / Г.Г. Киричек, О.О. Киричек // Науковий вісник Чернівецького національного університету. Комп'ютерні системи та компоненти. – Т. 2. – Вип.3. – 2011. – С. 14-21.

2. Обзор автоматических детекторов плагиата в программах [Электронный ресурс].- Режим доступа: http://logic.pdmi.ras.ru/~yura/detector/survey.pdf.

3. Макаров В.В. Идентификация дублирования и плагиата в исходном тексте прикладных программ [Электронный ресурс]/ В.В. Макаров.- Режим доступа: http://lab18.ipu.ru/projects/conf2006/1/15.htm.

4. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка [Текст] / Дж. Рамбо, М. Блаха. – 2-е изд. – СПб.: Питер, 2007. – 544 с.: ил.

5. Киричек Г.Г. Модель оцінки плагіату програмного коду на основі системи контролю версій [Текст] / Г.Г. Киричек, О.О. Киричек // Східно-європейський журнал передових технологій.– Харків: Технологічний центр, 2012.– №2/2(56).– С.25-32.