Projet Proposal for Parametrized String Matching Implementation for Software Plagiarism Check

Supervised By : **Prof. Srinivasaraghavan G**

Team
amit.tomar
(MT2013008)

siddhesh.dosi
(MT2013150)

srinivas.r.vaidya
(MT2013152)

@iiitb.org

29 - March - 2014

1 Introduction

This project aims at developing a Parametrized String Matching Implementation for Software Plagiarism Check, that given a collection of files which contain code in some programming language, will show a set of possible duplications of parts of the code among these. Comparing pieces of software will require discounting comments (optional and language dependent), extra/blank lines and spaces, variable renaming etc. The theory of parametrized string matching will be used to implementat this project. System will have an easy-to-use UI for selecting files/folders and shall report the plagiarism related information (matches found) in the UI in a nice manner.

2 Functional Requirements

Requirement.No.	1.1
Input	File option
Output	User prompted to select files from multiple folder.
Processing	Populate the folder structure of file system.

Requirement.No.	1.2
Input	Browsing and selection of files and next button
Output	User is promted to enter code snippet to be ignored while processing
Processing	File path validation

Requirement.No.	1.3
Input	Check plagiarism
Output	Plagiarism related log is generated.
Processing	Generate parameterized suffix tree to check amount of plagiarism.

Requirement.No.	2.1
Input	Folder option
Output	User is prompted to select Folder.
Processing	Search for all the files in the selected folder and populate a list.

Requirement.No.	2.2
Input	Selection/Deselection of files from the populates list and next button
Output	User is promted to enter code snippet to be ignored while processing
Processing	File path validation

Requirement.No.	2.3
Input	Check plagiarism
Output	Plagiarism related log is generated.
Processing	Generate parameterized suffix tree to check amount of plagiarism.

3 Non - Functional Requirements

External interface requirements :

1. Shall be portable, across various hardware and software platforms.
2. Shall be scalable and reliable.
3. Shall be easy to use.

Performance requirements :

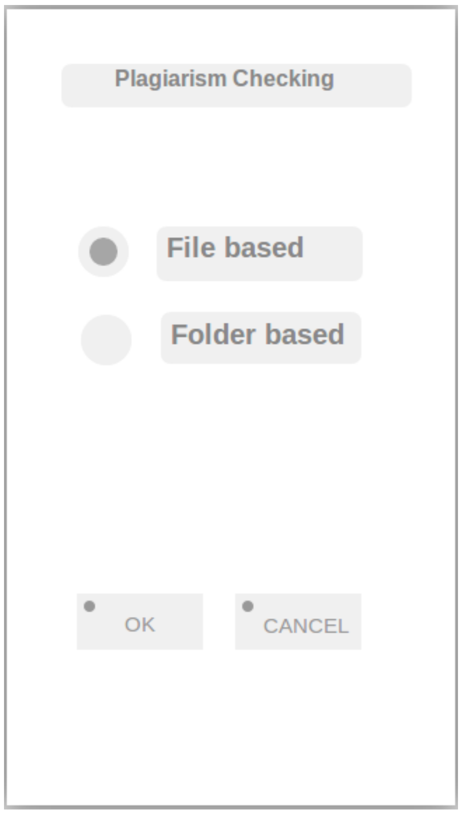
1. Shall have a good response time.

4 Goals of implementation

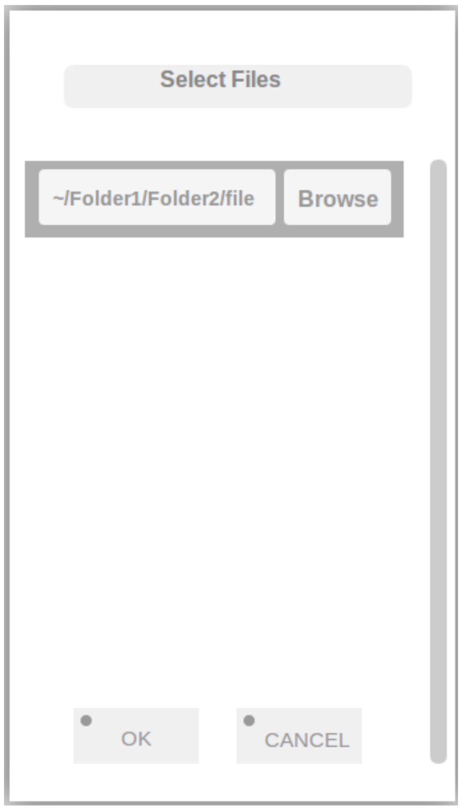
Plagiarism is a serious issue in computer science courses involving assessment of programming assignments [1]. Being electronic in nature, it is very easy to copy code and it is difficult to differentiate between the original and copied work. Thus, there is a need for a tool to detect plagiarism automatically, assisting professor to check for any kind of copying done by students.

5 UI Flow

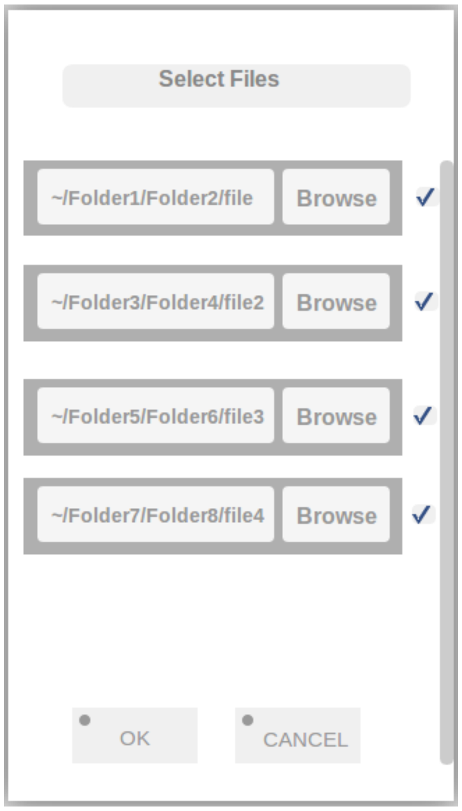
Following screen shots show the various UI screens :



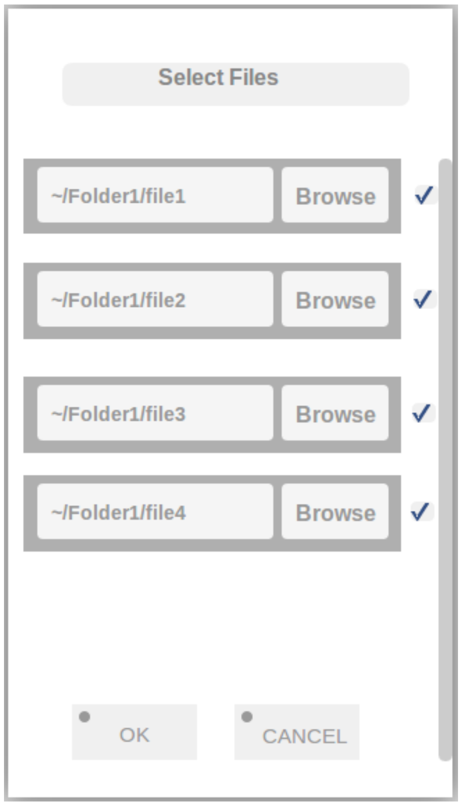
(a) Starting screen



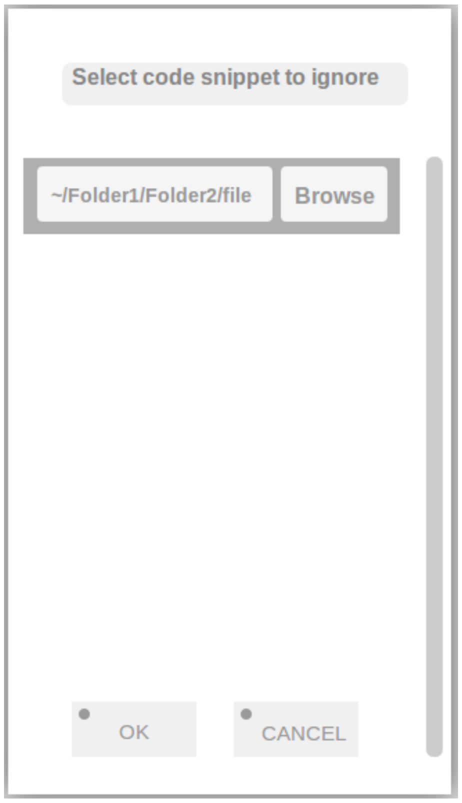
(b) Screen to select file



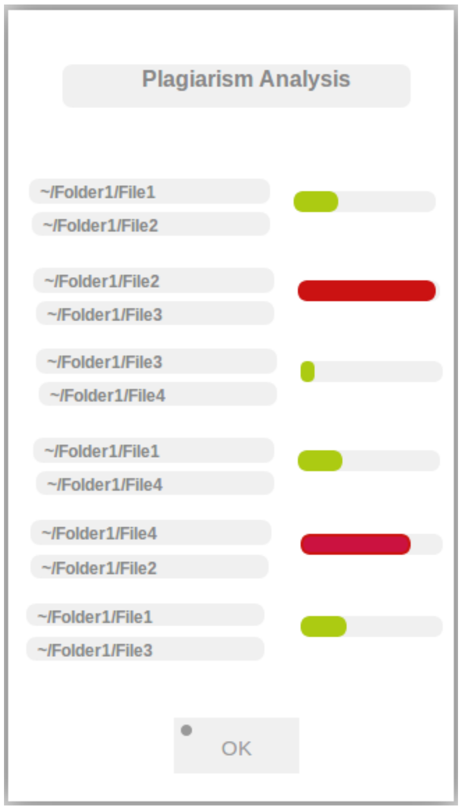
(c) Screen to select/deselect file



(d) Screen to display all the files in the selected folder



(e) Screen to select code snippet



(f) Screen to display plagiarism analysis report

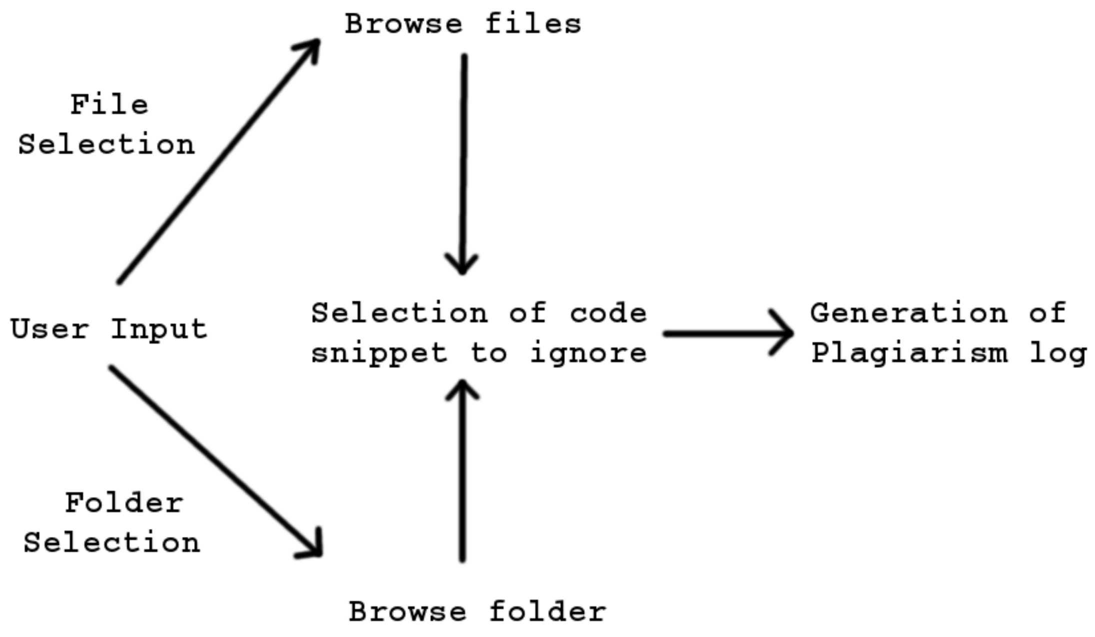


Figure 1: Decision Tree

6 Project deliverables and estimated time

1. **Milestones**

   (a) **Requirement Specification** :
   Date of submission : 7 - Feb - 2014
   (Already submitted to Prof. Srinivasraghvan)

   (b) **Literature Survey** :
   Expected date of completion : 15 - April - 2014
   Expected time : 40 Hours

   (c) **Building suffix tree data structure** :
   Expected date of completion : 15 - May - 2014
   Expected time : 30 Hours

(d) **Identifying duplicate code using suffix tree**:
Expected date of completion : 1 - June - 2014
Expected time : 25 Hours

(e) **Implementation of UI**:
Expected date of completion : 10 - June - 2014
Expected time : 10 Hours

(f) **Parameterized implementation for software plagiarism check**:
Expected date of completion : 1 - July - 2014
Expected time : 40 Hours

(g) **Integration of UI with parameterized string matching code**:
Expected date of completion : 15 - July - 2014
Expected time : 20 Hours

(h) **Testing**:
Expected date of completion : 31 - July - 2014
Expected time : 20 Hours

2. **List of final deliverables**:

   (a) Requirement specification document.
   (b) Design document.
   (c) User manual.
   (d) Deployment manual.

7 Hardware and Software requirements

S.No.	**Software**	**Version**	**Purpose**
1	Ubuntu Linux	13.04	Operating system.
2	GitHub	1.8.3.2	Version Control
3	Spyder	2.2.1	IDE for Python
4	GIMP	2.6	Image editing for documentation
5	Gummi	0.6.5	LaTeX editing for documentation

8 Coding guidelines

1. **Class**
   All class names should begin with capital letter, with all subsequent words beginning with a capital letter too.
   eg.
   class ThisIsAClassName
   class Administration
   class FooClass

2. **Function**
   All function names should begin with a small letter, with all subsequent words beginning with a capital letter.
   eg.
   void thisIsAFunction(void);
   int fooFunction (string) ;

   Names representing methods or functions must be verbs. eg.
   int getSystemVolume(void) ;
   void setSytemContrast(int) ;
   char findFirstCharacter(void) ;

3. **Variables**
   All primitive data type variables must begin with the first character of data type in small followed by the name of variable starting with a capital letter. Subsequent words will have their first character capital.
   eg.
   dataType dThisIsAVariableName
   int iSystemVolume;
   char cAnAlphabet;
   float fCurrentMonthSalary;
   long lVolumeOfWater;
   double dTotalTtax;
   string sMyName;
   bool bIsSet;

   Use of global variables should be avoided as far as possible.

4. **Objects of class** All object names must begin with ”obj” followed by the exact class name.
   eg.
   ```
   FooClass objFooClass;
   ```
   In case multiple instances of a class are to be used, above described name is to be followed by some information about the object. All these subsequent words must begin with a capital letter.
   eg.
   ```
   Employee objEmployeeAmit;
   Employee objEmployeeHemant;
   Employee objEmployeeKaustubh;
   ```

5. **Pointer types**
   If a variable is of pointer type, then its name should be preceded by ”ptr_”
   eg.
   ```
   int * ptr_iSalary;
   char * ptr_cNewCharacter;
   FooClass * ptr_objFooClassMemory;
   ```

6. **Array type** All array names should be preceded by ”arr_”
   eg.
   ```
   int arr_iSalary [ 10 ];
   char arr_cNewCharacter [ 200 ] ;
   FooClass arr_objFooClassMemory [ 5 ] ;
   ```

7. **List type**
   All list names should be preceded by ”lst_”
   eg.
   ```
   list[int] lst_iRollNumber;
   ```

References

[1] Peter Vamplew, Julian Dermoudy, ”An Anti-Plagiarism Editor for Software Development Courses”, *Proceeding ACE ’05 Proceedings of the 7th Australasian conference on Computing education - Volume 42 Pages 83-90* , 2005.

[2] E.M. McCreight ”A space-economical suffix-tree construction algorithm”, J. ACM 23,2 (1976), pp. 262-272.

[3] Brenda S. Baker, ”A Program for Identifying Duplicated Code, Computing Science and Statistics” 24 (1992), Interface Foundation of North America, pp. 49-57

[4] Brenda S. Baker, ”Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance”, SIAM Journal on Computing.