HÖGSKOLAN
Dalarna

On Occurrence Of Plagiarism In Published Computer Science Thesis Reports At Swedish Universities

Sindhuja Anbalagan

2010

Master Thesis
Computer Engineering

Nr:E4015D

DEGREE PROJECT
Computer Engineering

Programme **Master Programme in Computer Engineering- Applied Artificial Intelligence**	Reg number **E4015D**	Extent **15 ECTS**
Name of the student **Sindhuja Anbalagan**	Year-Month-Day **2011-02-06**	
Supervisor **Jerker Westin**	Examiner **Hasan Fleyeh**	
Company/Department **Department of Computer Engineering, Dalarna University**	Supervisor at Company/Department **Jerker Westin**	
Title **On Occurrence Of Plagiarism In Published Computer Science Thesis Reports At Swedish Universities**		
Keywords **Software clones , Software clones Detection, Clone Detection Methodologies, Plagiarism, Software Plagiarism Detection, Data Analysis, Ephorus, Minitab, Statistical Analysis**		

ABSTRACT

In recent years, it has been observed that software clones and plagiarism are becoming an increased threat for one's creativity. Clones are the results of copying and using other's work. According to the Merriam – Webster dictionary, "A clone is one that appears to be a copy of an original form". It is synonym to duplicate. Clones lead to redundancy of codes, but not all redundant code is a clone.

On basis of this background knowledge ,in order to safeguard one's idea and to avoid intentional code duplication for pretending other's work as if their owns, software clone detection should be emphasized more. The objective of this paper is to review the methods for clone detection and to apply those methods for finding the extent of plagiarism occurrence among the Swedish Universities in Master level computer science department and to analyze the results.The rest part of the paper, discuss about software plagiarism detection which employs data analysis technique and then statistical analysis of the results.

Plagiarism is an act of stealing and passing off the idea's and words of another person's as one's own. Using data analysis technique, samples(Master level computer Science thesis report) were taken from various Swedish universities and processed in Ephorus anti plagiarism software detection. Ephorus gives the percentage of plagiarism for each thesis document, from this results statistical analysis were carried out using Minitab Software.

The results gives a very low percentage of Plagiarism extent among the Swedish universities, which concludes that Plagiarism is not a threat to Sweden's standard of education in computer science.

This paper is based on data analysis, intelligence techniques, EPHORUS software plagiarism detection tool and MINITAB statistical software analysis.

ACKNOWLEDGEMENT

I begin with conveying my deepest gratitude to the Swedish government, for giving me an opportunity to pursue a Master's degree in Computer Science.

I would like to thank my supervisors Mr. Jerker Westin and Mr. Mevludin Memedi for their continuous support and guidance for my thesis work. They were very kind to me and approachable for any doubts and suggestions. Without their valuable suggestions, I guess this thesis would not be possible. I sincerely thank both of them, for their endless source of ideas and valuable hours of guidance for the past four months.

I would like to thank Mr. Hasan Fleyeh, Mr. Siril Yella, Mr. Pascal Rebreyend, and Mr. Mark Dougherty for their encouragements and valuable teachings.

Finally I would like to convey my deepest love to my parents and friends for their blessings and wishes.

TABLE OF CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

INTRODUCTION:

The objective of this paper is to review the methods for clone detection and to apply those methods for plagiarism detection and to analyze the results. Various detection methodologies will be analysed in detail.

Copying a code fragment and then reuse by pasting with or without alterations is a common activity in software development[1]. This type of reusing approach of the existing code is known as code cloning and the reused code fragment with or without alterations is called a clone of the original[2]. As a result of this reuse approach software systems often have a section of code which is similar, called software clones or code clones.

Cloning is often intentional [3] and are not necessarily harmful [4, 5] but there is a consensus that clones should be detected, in order to overcome the maintenance difficulties posed by such clones and to address a range of tasks that require the extraction of similar code fragments.

Several analyses show that, the software systems with code clones are more difficult to maintain, than the ones without them [6]. Cloning not only produces codes that are difficult to maintain, but also has a chance of introducing subtle errors [7]. Code clones are considered to be the bad smells of the software system today [8]. The maintenance life cycle of software systems, has been greatly affected by code clones.

Therefore in order to maintain the life cycle of software systems and to avoid the adverse effects of the software clones, it is beneficial to remove clones and prevent their introduction by continuous monitoring of the source code during its evolution [9].

There also cases in which clones are of special interest. For example, few software engineering tasks such as understanding code quality, aspect mining, plagiarism detection, software evolution analysis, copyright infringement investigation, virus detection or detecting bugs do require the extraction of similar code fragments.

Nevertheless, attempts are begin undertaken to detect clones [10] and when once the clones are identical then they can be removed through source code refactoring. Refactoring is kind of maintenance activity, but to note that "refactorings may not always improve the software with respect to clones" and "skilled programmers often created and managed code clones with clear intent"[11]

1.1 Terminologies:

The terminologies which are often used in the clone detection process are explained below:

Clone Pair:

A pair of code portions/fragments is said to be a clone pair, when there exists a clone relation (identical or similar) between the both.

Clone class:

A clone class is said to be maximal set of code portions/fragments, among which any of the two code portions/fragments hold a clone relation between them (i.e.) they tend to form a clone pair.

Clone class family:

The group of all clone classes that have the same domain is known as clone class family [12].

1.2 Types of clones:

The following types of clones are discussed in order to find the textual similarity [13], (i.e.) in order to find the occurrence and extent of plagiarism among the Swedish universities.

Type 1:

Identical code fragments except for variations in layout, white space and comments.

Type 2:

Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

Type 3:

Copied fragments with further modifications such as changed, added, removed statements, in addition to variations in literals , types, whitespace, layout and comments.

Functional Similarity:

When the functionality of the two code fragments is identical or similar, then it is known as semantic clone or type 4 clone.

Type 4:

If two or more code fragments, which perform the same computation, but are implemented by different syntactic variants, are known as type 4 clones.

2. CODE DUPLICATION

2.1 Reasons for code duplication:

Code clones do not occur by themselves either it could be developed or created accidentally. There are number of factors which could give rise to code clones. Some of those factors are,

(1) Cloning by accident.
(2) Clone occurrence during the development stage of the code.
(3) Cloning due to the programming techniques.
(4) Cloning for the system maintenance benefits.
(5) Cloning to improve the system efficiency in order to overcome the limitations.

The detailed description of each of these factors is as follows:

Cloning by accident:

Accidental cloning gives rise to more look alike than clones, which means that, when two developers were involved in implementing the same kind of logic and eventually come up with the similar procedures independently.

There also chances in which the developer’s apply the same kind of logic for similar problems , unknowingly. Those types of clones are the results of the side effect of developer’s memory.

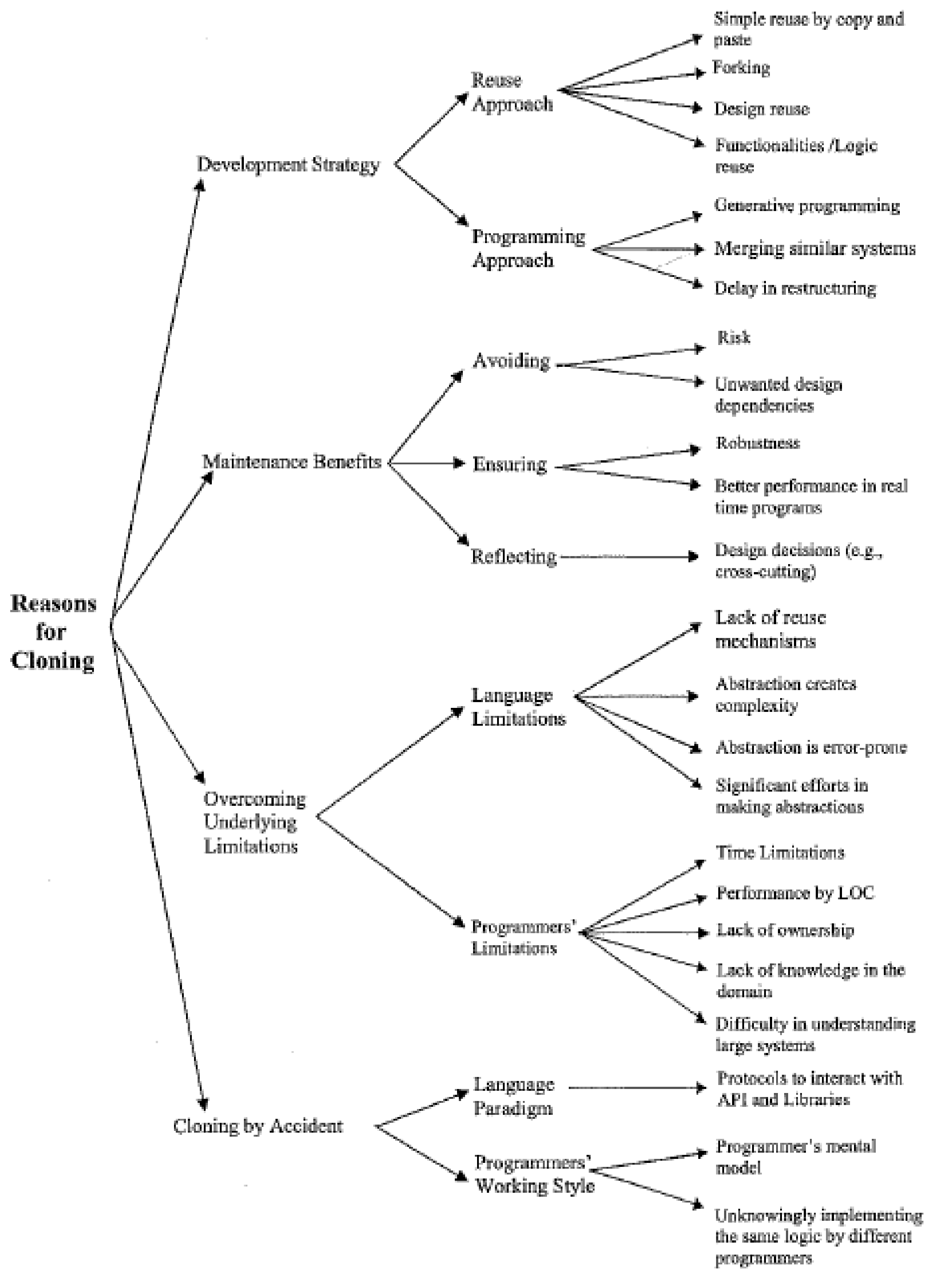

Reasons for Cloning
Development Strategy
Reuse Approach
Simple reuse by copy and paste
Forking
Design reuse
Functionalities /Logic reuse
Programming Approach
Generative programming
Merging similar systems
Delay in restructuring
Maintenance Benefits
Avoiding
Risk
Unwanted design dependencies
Ensuring
Robustness
Better performance in real time programs
Reflecting
Design decisions (e.g., cross-cutting)
Overcoming Underlying Limitations
Language Limitations
Lack of reuse mechanisms
Abstraction creates complexity
Abstraction is error-prone
Significant efforts in making abstractions
Programmers' Limitations
Time Limitations
Performance by LOC
Lack of ownership
Lack of knowledge in the domain
Difficulty in understanding large systems
Cloning by Accident
Language Paradigm
Protocols to interact with API and Libraries
Programmers' Working Style
Programmer's mental model
Unknowingly implementing the same logic by different programmers

Figure1: Tree diagram for the reasons for the cloning [20]

Clone occurrence during the development strategy:

There are more possibilities of clones during the design of the system. During the development stage, the developers are more interested in copying and pasting the existing code (i.e.) reusing the existing code. This is one of the simplest and fastest way of introducing a clone into the system. The term called “forking” is used by Kapser and Godfrey [14], which means the reuse of similar solutions, with the expectation that they will diverge significantly as the system evolves (i.e.) clones will be reused with slight modifications over time.

Clones due to the programming technique:

Merging of two software systems with some functionality, may give rise to clone .Though the two systems were developed by different teams, merging will result in the implementation of similar functionalities in both the systems, which could be considered as clone. Clones may also arise when there is a delay in restructuring the developed code. Generative programming approach often gives rise to huge code clones, because the same template is used in that tool to generate the same or similar logic.

Cloning for system maintenance benefits:

It is always easier and safer to use the existing successful code , than in attempt to develop a new code .The system maintenance is improved by using a well tested code, because it avoids the monetary consequences of software error (since 70% of software effort in the financial domain is spent on testing). Clones are also introduced in the system to maintain the software architecture clean and understandable.

Cloning to improve the system efficiency in order to overcome the limitations:

The two important limitations in software systems are language limitations and programmer’s limitations.

Language limitations often arise, when the programming language does not have the sufficient extraction mechanism e.g. inheritance, generic types (called templates in C++) or parameter passing , as result, the developers are urged to repeatedly implement these as idioms. Those repeating activities may cause potentially frequent clones [15].

Programmer's limitations often arise when the programmer does not have a complete or sufficient knowledge about the domain he/she works. Then, in order to succeed the task, he/she is forced to copy or reuse other's work which may lead to clone. The productivity of the developer or the developed code relies mainly on the efficiency of the code, but in some cases productivity is misunderstood as the number of lines of code. Under such misassumption, the developer tend to increase the code length by reusing or copying, thus results in clones. Time constraint-deadline in developing a code may also tend the developer to reuse to code, so as to finish before the deadline.

2.2 Merits of code duplication:

(1) Refactoring the cloned code, will improve the quality of the source code.

(2) Code fragments which has been reused multiple time can be incorporated in a library, citing its reuse is potentially official, because the often used code fragments proves its significant usability.

(3) When the functionality of a cloned fragment is known, it is possible to get a clear picture of other files which contains the other similar copies of the fragment.

(4) Functional usage patterns of a fragment could be found, if all the cloned fragments of a same source fragment can be detected [16].

(5) Detecting the code clones helps in aspect mining for detecting cross cutting concerns, Cross cutting concerns are nothing but the aspects of a program which affect other concerns. These concerns cannot be easily separated from the rest of the system in both the design and implementation, and could result in code duplication, tangling (significant dependencies between systems) or both.

2.3 Demerits of Code duplication:

(1) Code duplication threatens the safety of one’s idea or knowledge.

(2) Since, the existing well tested code is reused again and again, leads no way to analyze the problem in different approach.

(3) When the code that has been reused is not properly tested, it will serially affect all the segments, which have been using the code segments containing a bug.

(4) Cloning will affect the system design, when the reused code lacks in good inheritance and abstraction.

(5) When a cloned segment contains a bug, all of its counterparts should be investigated for the presence of the bug. This will increase the maintenance cost.

2.4 Applications of Clone detection:

(1) Clone detection is useful in finding malicious software. Malicious software is designed to secretly access a computer system without the owner’s informed consent. By comparing one malicious software with another, it is possible to find the matched parts of one software system with another [17].

(2) Clone detection helps in detecting plagiarism [17].

(3) Copyright infringement is known to be unauthorized or prohibited use of work under copyright ((i.e.) right to reproduce or perform the copyrighted work or to make derivative works) infringing the copyright owner’s exclusive right. Code duplication helps in finding in copyright infringement [18].

(4) As the time evolves, different clones in different versions get evolved, thus it helps in software evolution research [19].

(5) Clone detection techniques are used for compact device by reducing the source code size [20].

3.DETECTION PROCESS

3.1 General Clone detection process:

A clone detector must able to find the pieces of code, which is of high similarity in a system's source text. The main issue is that it is not aware beforehand which code fragments can be find multiple times. The detector essentially has to compare every possible fragment with every other possible to detect clones. Such comparison is more expensive in computational point of view, thus various measures are to be carried out in order to reduce the domain of comparison before performing the actual comparison. Once, potential cloned fragments are identified, and then further analysis is carried out to detect actual clones. The following is the overall summary of the clone detection process [20]

Preprocessing:

Initially, in the clone detection process the target source is partitioned and the comparison domain is determined. The main objectives to be considered in this phase are removing uninteresting parts, determining the source units and determining the comparison unit/ granularity.

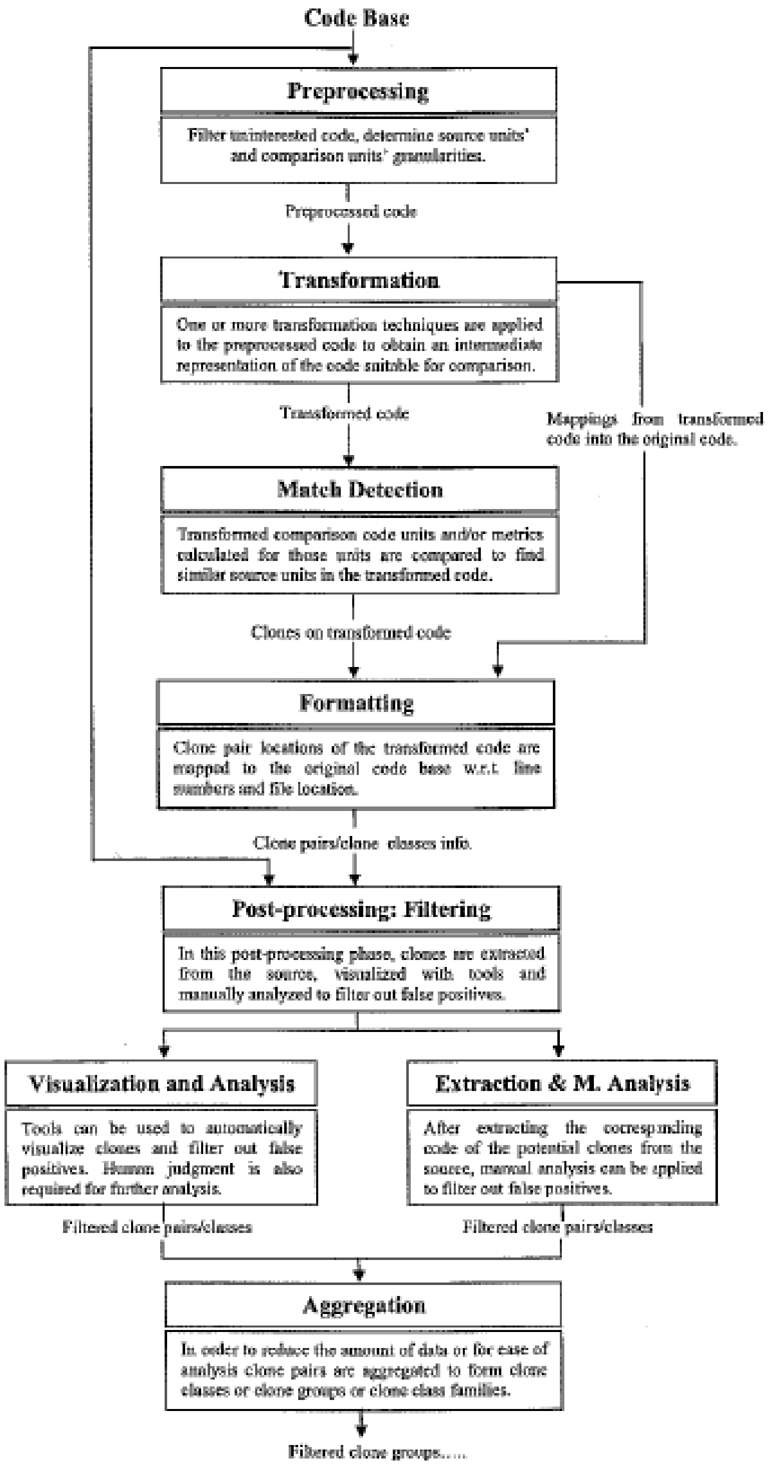

Code Base
Preprocessing
Filter uninterested code, determine source units' and comparison units' granularities.
Preprocessed code
Transformation
One or more transformation techniques are applied to the preprocessed code to obtain an intermediate representation of the code suitable for comparison.
Transformed code
Mappings from transformed code into the original code.
Match Detection
Transformed comparison code units and/or metrics calculated for those units are compared to find similar source units in the transformed code.
Clones on transformed code
Formatting
Clone pair locations of the transformed code are mapped to the original code base w.r.t. line numbers and file location.
Clone pairs/clone classes info.
Post-processing: Filtering
In this post-processing phase, clones are extracted from the source, visualized with tools and manually analyzed to filter out false positives.
Visualization and Analysis
Tools can be used to automatically visualize clones and filter out false positives. Human judgment is also required for further analysis.
Extraction & M. Analysis
After extracting the corresponding code of the potential clones from the source, manual analysis can be applied to filter out false positives.
Filtered clone pairs/classes
Filtered clone pairs/classes
Aggregation
In order to reduce the amount of data or for ease of analysis clone pairs are aggregated to form clone classes or clone groups or clone class families.
Filtered clone groups.....

Figure2: General clone detection process [20]

Transformation:

In the next stage, the source code's comparison unit is transformed to another intermediate internal representation for ease of comparison or for extracting the comparable properties. The transformation could be very simple by just removing the white space and comments [21] or could be very complex by generating PDG representation [22] or extensive source code transformations [17]. Few transformation techniques are pretty printing of source code, removal of comments, removal of whitespace, tokenization, parsing, generating PDG, normalizing identifiers, transformation of program elements and calculating metric values etc.

Match Detection:

The transformed code is given as input to a suitable comparison unit, where it is compared with each other in order to find the match. The order of comparison units are used, to sum up the adjacent similar units to form larger units. The output of the comparison unit is a list of matches with respect to the transformed code. These matches can be either the clone pair candidates or they have to be aggregated to form clone pair candidates. Then every clone pair is generally represented with the location information of the matched fragments in the transformed code.

Formatting:

In this stage, the clone pair list obtained with respect to the transformed code is then converted to a clone pair list obtained with respect to the original code base. Normally, after finding the clone pair location from the previous phase, it is then converted into line numbers on the original source files.

Post processing:

This stage helps out in filtering the false positive in two ways such as manual analysis and visualization tool.

Manual Analysis:

Once the original source code has been extracted, the raw code of the clones of the clone pair is subjected to the manual analysis, in order to filter out the false positive.

Visualization:

To speed up the manual analysis in filtering out the false positives, visualization tool are used to visualize the clone pair.

Aggregation:

Finally, to perform certain analysis, we have to reduce the amount of data, so the clone pairs should be aggregated to clusters, classes, cliques of clones or clone groups etc.

4.DETECTION TECHNIQUES

This chapter discusses about the six clone detection techniques. They a

1) Text Based Technique.
2) Token Based Technique.
3) Tree Based Technique.
4) PDG Based Technique.
5) Metric Based Technique.
6) Hybrid Techniques.

They are discussed in detailed as below.

4.1 TEXT BASED TECHNIQUES:

Pure text based/string based methods are incorporated in several clone detection techniques today. In this approach, the target source program is assumed to be sequences of lines/strings. And then, two code fragments are compared with each other to find the matched sequences of text/strings. When a match is found ((i.e.) two or more code fragments are found to be similar), then they are returned as clone pair or clone class by the detection technique.

As the name says, it is purely based on textual/lexical approach, so the detected clones do not correspond to structural elements of the language. Mostly, the raw source code is directly used in clone detection process, without any transformation/normalization. However, few of the latest text based clone detection technique use some of the transformation/normalization such as comments removal, whitespace removal etc.

Algorithm Used:

1) Line based string matching algorithm

2) Suffix tree algorithm.

Detection tool:

The popular detection tool used in this method is DUP.

DUP :

➔ Supporting Language	:	C, C++, Java.
➔ Domain	:	CD/UNIX.
➔ Approach	:	Line based/Text based.
➔ Background	:	Academic.
➔ Validation	:	With two Systems

DUP Detection Tool

DUP finds either maximal exact matches over a user defined threshold length or maximal parameterized matches over a user threshold length. The threshold length is the smallest clone size that DUP will report, measured in terms of noncommentary source lines(ncsl) in each fragment.

DUP does not parse the input, but it tokenizes the input and classifies each token as a parameter or non parameter. Tokenization depends on the language being processed. Identifiers and constants are assumed as parameters and keywords, punctuation are assumed as non parameters.

A string of tokens , also known as symbols is known to be parameterized string or p-string. Two p-strings are known to be exact match if they contain exactly the same symbols. Two p-strings are known to be parameterized match(p-match) , if there exists a one to one function on a alphabet of parameter and non parameter symbols such that the function maps each non parameter symbol into itself and each parameter symbol into a possibly different parameter symbol and one string is mapped by this function into the other.

DUP Description:

The size of the clones detected by the DUP based on PDG technique is expressed as the number of vertices in the PDG that are included in a clone and in DUP based on the text based detection it is expressed as the number of lines of the source code. For the cloned code(cc) component or fragment , three vertices is the smallest minimum size that could still be handled, because two vertices cause PDG to abort.

DUP requires the user to set a commonality threshold which is used to remove clones that are overlapped too much by other clones. It was suggested , the common threshold was set to 80%. DUP able to match the annotated crosscutting concern code. Originally few blank lines and lines containing only opening and closing brackets (i.e.) "{" and "}" were included in the annotations. Those lines will never be included in the results of DUP clone detector because such lines are not included in the PDG-DUP's mapping from PDG vertices to source code or during transformation. Hence such lines had their annotations removed.

For Memory handling , the DUP clone detector performs the best. The recall precision for DUP is significantly above those of CCdiml and CCfinder clone detectors. DUP is bound to conform to language syntax as well since its PDG's are built on top of AST's. DUP results in higher level of precision, by making a better distinction between, the code that is similar at the lexical level, yet conceptually different.

DUP lacks efficiency in Error handling. DUP clone classes have the best match with the tracing code. DUP performs the best in NULL value checking. A limitation that is surfaced mainly for the

DUP detector is due to the line granularity of the case study. Each source code line can belong to atmost one concern while , in some cases, we could consider including a line in multiple concern. The DUP clone detector appears to be most suitable for such refactoring activities since it respects both syntactic integrity and include context dependencies in its clone.

DUP options and performance:

DUP allows the user to set the threshold size for reporting maximal matches. DUP is used to find maximal p-matches or only maximal exact matches, or when a p-match is also an exact match. It has an option to filter out p-matches that have too many changed parameters. Since the number of p-matches found in large software system is often large, the filtering option can be used to eliminate some of the less useful matches.

DUP is implemented in about 4000 lines of C and runs under UNIX. It is very fast. For example, on the largest project analyzed in the experiment , with 202k source lines and 136k noncommentary source lines , DUP used 11.7 seconds and 57MB when run on one processor of an IRIX machine with eight 250Mhz processors.

Initially some detectors where based on lexical analysis. For instance, Baker's dup [21, 23] used sequence of lines, to represent a source code and line by line clone checking was done. Therefore, lexer/line based algorithm was incorporated.

Dup tool removes whitespaces, tabs and comments. The identifiers of functions, variables and types where replaced by a special parameter. It concatenates all lines which have to be analyzed into a single text line. It hashes each line for comparison. Using suffix tree algorithm, it extracts a set of pairs of longest matches

Some Examples:

Baker's Approach:

Baker[1] also analyzed the problem of finding exact and near duplication in software. In this approach, Baker investigated the objective of parameterized matches (p-match) in Dup tool. A code fragment is said to be matched (clone), if both the fragments are contiguous sequences of source lines with some consistent parameter/ identifier mapping scheme.

Johnson's Approach:

Johnson's [6] redundancy finding mechanism using fingerprints on a substring of the source code is another approach in pure text based approach. Karp- Rabin fingerprinting algorithm is used. In this algorithm, signatures to be analyzed are first collected, then signatures are calculated per line are compared in order to detect the matched substring.

This algorithm is used for calculating the fingerprint of all length n substrings of a text. Initially, a text to text transformation is carried out on the source file, so as to discard the uninterested characters. Then the whole text is subdivided into set of substring, so that every character in the text appears in atleast one substring.

Finally, the matched substrings are identified. The results obtained could be further improved, by applying a transformation. Instead of applying a set of text to text transformation, Johnson applied a transformation technique which is a combination of all basic transformation.

Advantages:

(1) Most widely used technique.

(2) Easy to implement.

(3) To eliminate noise, it is possible to remove uninteresting language elements.

Disadvantages:

(1)There are several problems could be arised in a line by lin technique, they are line break, identifier changes, parenthesis removal/adding for a single statement and transformation.

(2)Dup tool does not support exploration and navigation through the duplicated code.

(3)Detection accuracy is low.

4.2 Token Based Technique:

In this approach, the entire raw source code is lexed/parsed/transformed into sequence of tokens. Then this sequence of tokens is scanned for finding the duplicated sequence. At last, the original code portions which represent the duplicated sequences are returned as clones.

Algorithm Used:

Suffix tree based sub string matching algorithm.

Detection tool:

The clone detection tools used in this method are *CC* Finder, RTF, CP-Miner.

The plagiarism detection tool used in this technique are winnowing, JPlag and SIM.

***CCF*inder:**

➔ Supporting Language	:	C,C++, Java, COBOL.
➔ Domain	:	CD / Windows / NT.
➔ Approach	:	Transformation/Token based technique.
➔ Background	:	Academic.
➔ Validation	:	With four Systems.

RTF:

➔ Supporting Language	:	C.
➔ Domain	:	CD.
➔ Approach	:	Token / Suffix array.
➔ Background	:	Academic.
➔ Validation	:	With Linux parts.

CP-Miner:

➔ Supporting Language	:	C, C++.

➔ Domain	:	CD / Windows / Linux.
➔ Approach	:	Sequence Database/ Frequent subsequent.
➔ Background	:	Academic.
➔ Validation	:	Several Systems.

JPlag :

➔ Supporting Language	:	C, C++, Java, Scheme, NL text.
➔ Domain	:	PD / Online.
➔ Approach	:	Token / Greedy.
➔ Background	:	Academic.
➔ Validation	:	With two Systems.

SIM :

➔ Supporting Language	:	C
➔ Domain	:	PD / Linux.
➔ Approach	:	Parse tree to string / String alignment.
➔ Background	:	Academic.
➔ Validation	:	With several Systems.

Example:

Kamiya et al Approach:

This approach was one of the leading states, in the token based technique, which employed CC Finder detection tool [17]. Then each line of the source is partitioned into tokens by lexer, and tokens of all the source file are concatenated to a single token sequence. Transformation (i.e. tokens are added, removed or changed) on the basis of the transformation rules of the language interest is applied, which aims at regularizing the identifiers and identifying the structures. Each identifier in relation to their types, variables and constants are replaced with a special token. Thus this identifier replacement enables code fragments with different variable names clone pair.

A suffix tree based sub string algorithm is applied to detect the similar sub sequence on the transformed token sequence, once when the matches are found, the similar sub sequence pairs are returned as clone pairs/clone classes. As the clone pair/clone class information is obtained with respect to the token sequence, a mapping is required to obtain the essential clone pair/clone class information with respect to the original source code.

Advantages:

(1) This approach is usually more robust against code changes such as formatting and spacing, when compared to text based approaches.

(2) Fragile to statement reordering and code insertion due to the sequential analysis in *CC*Finder and Dup.

(3) Generally, a reordered or inserted statement can break the token sequence, which could be later disregarded as duplicate to another sequence. This limitation can be overcome, by using CP Miner, which employs frequent subsequence mining technique, where a frequent sequence can be interleaved in its supporting sequence.

Disadvantages:

(1)Sometimes noise arises as a result of suppression of insignificant token classes (e.g. Access modifiers of Java).

4.3 Tree Based technique:

In this technique the program is pared into a parser tree or an abstract syntax tree (AST) with a parser of language of interest. Then, using a tree matching technique, similar sub trees are searched in the tree, when a match is found corresponding source code of the similar sub trees are returned as clone pairs or clone classes. The complete information about the source code is available in the parse tree or AST. The variables names and literal values of the source code are discarded during the tree representation; still it is possible to employ more sophisticated clone detection tools.

Algorithm used:

Tree matching Algorithm [20].

(1) An annotated (AST) parse tree is generated by a complier generator.
(2) Using tree matching, AST compares it sub trees by characterization metrics based on a hash function.
(3) When a match is found source code of similar trees are returned as clones.

The hash function is used to detect the parameterized matching, to detect the gapped clones and also to identify the code portions in which some statements are reordered.

Detection Tools:

There are two detection tools used in this technique, CloneDR and ccdiml.

CloneDR :

➔ Supporting Language	:	C, C++, Java, COBOL.
➔ Domain	:	CD, windowNT.
➔ Approach	:	AST / Tree Matching.
➔ Background	:	Commercial.
➔ Validation	:	Process Control System

Ccdiml :

➔ Supporting Language	:	C, C++
➔ Domain	:	CD / Linux
➔ Approach	:	AST / Tree Matching.
➔ Background	:	Academic.

➔ Validation : Against several systems.

Ccdinl differs from CloneDR , by having an extra advantage like the avoidance of the similarity metric, in handling the sequence and hashing.

CloneDR differs from ccdinl, by working concurrently, in order to maintain and check the consistency.

Some Examples:

Yang Approach:

Yang's approach [24] aimed at finding the syntactic differences between two versions of the same program. In his approach, he generated a variant of parse tree for both the versions, and he applied dynamic programming in searching similar sub trees.

Wahler et al Approach:

Whaler and others [25] aimed at detecting the exact and parameterized clones in a more abstract level than AST. In his approach, first he attempted in converting the AST of the program into XML. Then he employed data mining frequent item set technique to the XML of the AST in order to find the clones.

Evans and Fraser Approach:

Evans and Fraser [26] proposed a method for structure abstraction. They aimed at finding the exact and near miss clones with gaps. In their approach, they built ASTs from lexical abstraction of a program by parameterzing only AST leaves (i.e.) abstraction of identifiers and literal values. Then by further parameterizing the arbitrary sub trees of ASTs structural abstraction is obtained.

Advantages:

(1)Clones can be determined in linear time and space by using current techniques based on abstract syntax suffix trees.

Disadvantages:

(1)ASTs are fragile to statement reordering and control replacement, because they disregard the information about identifiers and ignores the data flow.

(2)No special treatment for identifiers and literal values in ASTs for detecting clones.

These limitations could be overcome by using PDG based technique.

4.4 PROGRAM DEPENDENCY GRAPH (PDG) TECHNIQUE:

Program dependency graph (PDG) approach considers the semantic information of the source code, so it goes one step further in obtaining the source code representation of high abstraction than any other approach. Semantic information is carried in PDG, because it contains both control flow and data flow information of a program. When a set of PDGs are obtained from a subject program, isomorphic sub graph matching algorithm is employed , in order to find the similar sub graphs, which are then returned as clones.

Algorithm:

Isomorphic Sub graph matching algorithm.

Detection Tool:

The tools that are based on PDG techniques are,

Clone detection tool, PDG –DUP

Plagiarism detection tool, GPLAG

PDG-DUP:

- Supporting Language : C,C++
- Domain : CD / Clone refactoring
- Approach : PDG / Slicing.

➔ Background	:	Academic.
➔ Validation	:	With some systems

CCFinder:

➔ Supporting Language	:	C, C++, Java, COBOL.
➔ Domain	:	CD / Windows / NT.
➔ Approach	:	Token based technique.
➔ Background	:	Academic.
➔ Validation	:	With four Systems.

Examples:

Komondoor and Horwitz's Approach:

This approach [22, 27] aims at finding the isomorphic PDG subgraphs using program slicing. This is the most leading PDG based clone detection approach, but they also proposed an approach to support software refactoring, in which they group identified clones together while preserving the semantics of the original code for automatic procedure extraction.

Krinke Approach:

This approach [28] uses iterative k-length patch matching for detecting maximal similar sub graphs.

Chen at al. Approach:

Chen at al and others [19] proposed a technique using PDG for code compaction which has several advantages in embedded systems. This approach has concern both in syntactic structure and data flow.

Advantages:

PDG-based techniques are more robust to reordered statements, insertion and deletion of code, intertwined code, and non-contiguous code.

Disadvantages:

This technique is not scalable to large size programs.

4.5 Metric based technique:

In this technique, instead of comparing the code directly, different metrics of code were gathered and these metrics were compared to detect the clones. Many clone detection techniques today uses software metrics for detecting similar codes. Initially, fingerprinting functions which are nothing but a set of software metrics are calculated for one or more syntactic units such as a class, a function, a method or even a statement and then these metric values are compared to find clones over these syntactic units. Generally such metrics are calculated by parsing the source code into AST/PDG representation. Then the metrics were calculated from names, layout, expression and simple control flow of functions. A clone is detected only when pair of whole function bodies that have similar metrics values are identified.

Detection Tool:

PDG-DUP:

➔ Supporting Language	:	C, C++
➔ Domain	:	CD / Clone refactoring
➔ Approach	:	PDG / Slicing.
➔ Background	:	Academic.
➔ Validation	:	With some systems

Examples:

Mayrand et al

This approach [29] considered each function units of a program for calculating the metrics, say for example, number of lines of source, number of function calls contained, number of CFG edges, etc. Code clones are identified to be units with similar metric values. Partly similar units were not detected. Intermediate Representation Language (IRL) was used to represent the source, which characterized each function in the source code.

Kontogiannis et al Approach:

In this approach [30], probable matches were identified by building an abstract pattern matching tool using Markov model. Initially this approach was used to find the similarities of the program alone, but not find the copy-pasted code. Later, Kontogiannis suggests two ways to find clone. The first way deals with the direct comparison of the metrics values that classify a code fragment in the granularity of begin − end blocks, assuming that two code fragments are similar if their corresponding metrics values are proximate. The second way uses a dynamic programming technique for comparing begin − end block in a statement-by-statement basis.

Lanubile and Calefato Approach:

This approach [31] proposed a semi-automated method for detecting cloned script functions. In which, potential function clones were detected with an automated approach and then a visual inspection was employed in the selected script functions. eMetric tool were employed to retrieve the potential function clones and based on the tool report visual inspection were carried out on the code of the selected script functions , in order to detect suspect clones.

Advantages:

(1)Similar static HTML pages could be identified by computing the distance between items in web pages and evaluating their degree of similarity.

Disadvantages:

(1)This method needs a special tool to generate the potential clones associated metrics.

Human attention is needed for clone refactoring.

4.6 Hybrid Approaches:

Few cases need to explore the advantages of one or more techniques in order to detect the clone, and then hybrid approach is employed. Hybrid code representation and/or technique is used by several other detection approaches in detecting clones. This approach is nothing but the combination of previous discussed techniques.

Detection Tool:

DUP:

➔ Supporting Language	:	C,C++, Java.
➔ Domain	:	CD/UNIX.
➔ Approach	:	Line based/Text based.
➔ Background	:	Academic.
➔ Validation	:	With two Systems

CC*F*inder :

➔ Supporting Language	:	C, C++, Java,COBOL.
➔ Domain	:	CD / Windows / NT.
➔ Approach	:	Transformation/Token based technique.
➔ Background	:	Academic.
➔ Validation	:	With four Systems.

Examples:

Koschke et al. Approach:

This approach [32] was based on token and tree based techniques. This aimed at finding the syntactically similar sequence, this was carried out by serializing the AST nodes in the preorder traversal, a suffix tree was created for these serialized AST nodes.In resulting maximally long AST node sequences , syntactic regions were cut so that only syntactically closed sequences alone remain. This approach compares the tokens of the AST-nodes using a suffix tree-based algorithm, instead comparing the AST nodes. Thus, this hybrid combination allows one to detect the clones in linear time and space than the conventional AST-based approaches.

Microsoft's new Phoneix Framework:

A function-level clone detection technique was proposed for the Microsoft's new Phoenix framework based on AST and suffix trees [33]. As in Koschkeet, AST nodes were used to generate a suffix tree, which enables the nodes to detect the clones linear time and space. This approach also able to detect the exact matching function clones and parameterized clones with identifier renaming. Greenan proposed a similar approach for finding level clones on transformed AST using sequence matching algorithm.

Balazinska et al. Approach :

This hybrid approach [34] was proposed for characterization of metrics and dynamic pattern matching. Using metric based approach, characteristics metric values were calculated for each method bodies. Then using Dynamic pattern matching, token sequence for each pair of similar methods were compared in order to identify clones. Thus this a hybrid approach of metric and token based techniques.

Advantages:

(1)Hybrid approach can also be used for LISP – like languages.

(2)This approach has an additional advantage, that each of its parts is replaceable.

(3)By specifying their syntax, new languages can be added.

(4)By including new specialized comparison functions, it is possible to explore other language features.

Disadvantages:

(1)This approach is not fully language independent, it needs parser, classification algorithm and metrics generator.

5.MEASURES FOR AVOIDING CODE DUPLICATION

5.1 Avoidance of clones:

Clones are regarded as harmful agents in software maintenance, so it should be detected and removed. It would be even better if there is no clone at all, so that we need not to think either about detection or removal. The best way to avoid clones in order to maintain clean and efficient software is to use the preventive approaches or clone detection tool right from the beginning.

There are two preventive measures [9] in the development process to avoid clones. They are

(1) Preventive Control
(2) Problem mining.

Preventive Control:

Whenever adding a new function to a system, it has to be checked well to confirm that, this new function is not a clone to an existing one or there should be any specific reasons for adding that new function as a clone to the system.

Problem mining:

In this any modification to a function must be consistently propagated to all of its similar functions is a system, so, it reduces the chance of creating clones unnecessarily and the probabilities of update anomalies are reduced significantly.

5.2 Removal of Code Clones :

When the preventive measures to avoid the clones fails, clones occur in the system. Then the next issue, is to remove the clones. Removal of clones from the software system is done through refactoring. Using clone refactoring it is possible to decrease the complexity, to reduce the potential source of errors occurring from the duplicated code.

Exact method refactoring:

This is simplest way in refactoring of clones [35]. This method, replaces the cloned code by a function call, to the newly created function which is created from the shared code of clone fragments. Clones which resulted from exact copies or differ only in identifiers are possible for such simple functional abstraction.

Synytskyy and others used traditional reuse techniques of dynamic websites to find and resolve the clones that occur in web documents. They used a multi pass approach for resolving clones incrementally, using various different resolution techniques, resolving each clone encountered with the most suitable resolution method available.

Clones tend to be refactored generally during the evolution of software. Refactoring of clones is risky and most potential source of errors. For this reason, code developers are always reluctant in code refactoring. As an alternative, code refactoring can be optimized with constraints and conditions applied during refactoring.

5.3 Management of Clones:

Management of clones in a software system stands far better than removal or avoidance. Code refactoring is always not possible or practical for some existing clones. Sometimes it is not cost effective too. But it is always possible to manage or keep track of clones either in their individual version or in the evolving version of the system.

Simultaneous Editing:

This is the first attempt towards the management of clones [36]. The repetitive text editing task of the system is simplified by this approach. Users provide the regions to be linked in repetitive text records through selection or through specifying a text pattern. Whenever there is an edit made to any of the linked records, the user can able to see equivalent edits which are applied simultaneously to all other records.

The shape of the code varies depending on the variation of the shape of organization. Hence, it is always beneficial to identify and understand the patterns of the developing team that deals with the duplicated code. These patterns gives better understanding about both the project structure and its developing team, thus helps in better management of clones.

6.PLAGIARISM

6.1 Introduction to Plagiarism:

Plagiarism is an act of taking the credit of someone else work. It is also known as the unauthorized use or close imitation of the language and thoughts of other author and the representation of those as one's own work.

6.2 Detailed Definition:

As stated earlier, it is an act of taking credit of someone's work. In college/universities, this much involves in writing, but other form of works can also be plagiarized, such as music, ideas and artwork. Taking credit of someone's work is stealing and it is a violation of Intellectual Property Law. So plagiarism is more than a just violation of teacher's trust and school policies. It is an illegal activity which is not so different from stealing one's iPod or wallet.

What actions are considered plagiarism?

In any time, someone uses another author's words or thoughts, without giving a proper credit to them .Here are some examples [37],

Directly quoting another person's paper:

This is the most obvious example, happens all the time. It could be from college essay plagiarism website that buy and sell term papers, or from a friend.

Directly quoting someone else's phrase:

Instead of stealing the whole paper one could steal few paragraph or sentence from a book or website. If the credit to the original author is not mentioned, then it is known as plagiarism.

Using someone else's idea:

This mostly happens in academic work. Usually, the idea's from the fellow students are often stolen and used by others during labs and projects. This is still considered to be plagiarism.

Recycling your old material:

In some cases, a student wants to expand upon an idea from another paper in another class, this is not considered to be plagiarism, unless, he/she attempts in tweaking the contents of one paper or assignment to meet the requirements of another paper or assignments.

Fail to acknowledge the source or failure to put the quote in the quotation mark:

These are probably not malicious and known to be sloppy errors. But still, in technical terms they are said to be plagiarized.

6.3 Reasons to avoid plagiarism:

(1) Plagiarism would arrest the discovery of new idea's and life time inventions, when there is no practice of thinking for themselves.

(2) One would completely fail to discover the power of their mind in the creation of interesting and new ideas.

(3) The greatest of achievement in receiving praise, would not be felt by anyone for their plagiarized work.

(4) There are more chances to be caught, embarrassed and punished.

(5) Though you have your name on your work, if it is plagiarized, then it is no more your unique work.

6.4 How to prevent plagiarism?

By enforcing strict laws and making plagiarism as a serious offense, this could be avoided. Maintaining an electronic database for previous paper, will be helpful for comparison, which in turn could prevent plagiarism.

Sincerely speaking, its one’s dedication, loyalty towards their work, respect for other’s idea and proper usage of references could prevent plagiarism.

7. SOFTWARE PLAGIARISM DETECTION

7.1 Manual Plagiarism detection:

The idea is to validate one’s work by comparing it with the all available sources. The detection mechanism becomes more complex when human hands are involved. The database available for comparison is extremely large and it is always not possible for anyone to explore them completely. Manual plagiarism detection is,

(1) Time consuming.
(2) Very low in efficiency,
(3) Lacks in accuracy.

For example,

Manual plagiarism detection in colleges can cross check the reports of the fellow students alone, but not against the all available sources. So, there are more chances of lack in efficiency.

So, in this paper Software plagiarism detection tool has been used.

7.2 Software plagiarism detection:

Software plagiarism detection is more efficient and accurate than manual plagiarism detection method.

Ephorus software [38] is used for detecting the percentage of plagiarism.

7.3 Ephorus:

In this paper, Ephorus anti-plagiarism software is used to detect the percentage and sources of plagiarism. Ephorus is based on text based detection method.

Let's look into detail, how does plagiarism detection works? What type of documents could it process? How could it process? What are the sources that are checked for possible plagiarism? How the results are evaluated?

Working of Plagiarism Detection:

Whenever a student submits their documents, teachers hand in the documents in the Ephorus software for plagiarism check. This is done as figure represented below [39]. Detailed explanation is given in the following part.

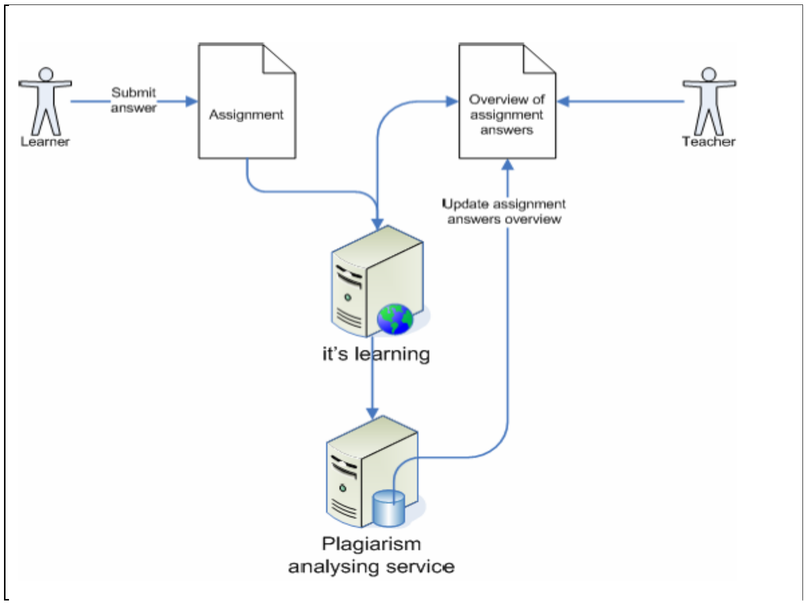

Learner
Submit answer
Assignment
Overview of assignment answers
Teacher
Update assignment answers overview
it's learning
Plagiarism analysing service

Figure 3: Working of Plagiarism detection[39].

Sources for plagiarism detection:

Ephorus will automatically compare all work handed into the system with [39]:

(1) Documents found on the Internet.

(2) Other documents handed in by fellow students.

(3) Documents handed in by other institutions using Ephorus.

7.4 Ephorus and its Clone detection tool:

Ephorus is based on JPlag tool.

JPlag:

JPlag is a system that finds similarity among multiple set of source code files. It employs text based detection approach,this way it can detect software plagiarism. JPlag does not merely compares bytes of text, but is aware of programming language syntax and program structure and hence is robust against many kinds of attempts to disguise similarities between plagiarized files. JPlag currently supports Java, C#, C, C++, Scheme and Natural language (NL) text. JPlag is typically used to detect and thus discourage the disallowed copying of student exercise programs in programming education.

But in principle it can also be used to detect stolen software parts among large amounts of source text or modules that have been duplicated(and only slightly modified). JPlag has already played a part in several intellectual property cases where it has been successfully used by expert witnesses. JPlag has a powerful graphical interface for presenting its results.

Allowed Document types:

Only the following documents could be handed in Ephorus for plagiarism detection [39].

FILE TYPE	ABBREVIATION
Microsoft word	*.doc
Rich text format	*.rtf
Htm	*.htm
Html	*.html
Adobe Acrobat	*.pdf
Open office	*.sxw
Plain text	*.txt

Table 1 : Allowed document types in Ephorus [39].

Allowed File size:

Ephorus accepts documents upto a maximum file size of 16MB.

ZIP files can have a maximum size of 4MB.

Possible check for plagiarism:

Default check:

Document will be checked for plagiarism, and used as reference material for future checks.

Reference material:

Document will not be checked for plagiarism, but saved as reference material for future checks.

Private check:

Document will be checked for plagiarism, but will not be used as reference material for future checks

Two types of Results :

Summary:

Gives the overall percentage of plagiarism and percentage level of various source contributions. The document in orange color is the original document and document in black color is possible plagiarism.

Detailed report:

The detailed report zooms in on the results and compares the submitted document and individual sources side-by-side.

Final results are calculated after cross checking the software results manually in order to avoid the false positives.

7.5 Other Software Plagiarism detection systems:

Turnitin Plagiarism prevention:

Turnitin is a plagiarism prevention tool from iParadigms.

iParadigms:

The iParadigm team is a group of dedicated professionals that includes award winning teachers, graphic designers, computer scientists and business professionals working together to stop the spread of internet plagiarism and promote new technologies in education.

Turnitin Description:

Turnitin allows educators to check students work for improper citation or potential plagiarism by comparing it against continuously updated database. Every originality report provides instructors with the opportunity to teach their students proper citation methods as well as to safeguard their institutions academic integrity.

Features and Benefits:

(1) Scans 3 extensive databases.
(2) 13.5 + billion web pages.
(3) 130 + million student papers
(4) Thousands of newspapers , magazines , books and scholarly journals.
(5) Encourage proper citation.
(6) Shows side by side comparison with colour coded matches.

Urkund Plagiarism detection system:

It is possible for a teacher , to check for plagiarism with Urkund plagiarism detection system.

Urkund Description:

Urkund is owned and developed by PrioInfo AB. PrioInfo is a company with over 25 years of experience of the requirements and needs of information intensive organization. PrioInfo is an agent for net based services from a multitude of International information providers and publishers. It also delievers a licensed ebook platform to corporations , publishers and libraries as well as universities.

Plagiarism control with Urkund:

Urkund deliver plagiarism control that checks learner assignments with the internet, published material and our archive of previously submitted student documents. A Urkund report presents the teacher with the information required to determine whether or not plagiarism is the case.

8.APPLICATION OF CLONE DETECTION TECHNIQUES

8.1 Plagiarism detection among Swedish universities:

Determining the extent of plagiarism among Swedish universities using Ephorus software.

In order to find the plagiarism among Swedish universities following steps were followed:

Step 1: Collecting Computer Science Master Thesis report (samples) across

Swedish universities .

Step 2: Processing the documents in Ephorus software to find the possible sources

of plagiarism.

Step 3: Cross checking the matched sources manually in order to avoid false

Positives which incorporates HUMAN INTELLIGENCE TECH IQUE.

Step 4: Once the percentage of plagiarism has been found, STATISTICAL

DATA ANALYSIS is carried out using Minitab software.

Each step has been in detailed has follows:

Step 1: Collecting Computer Science Master thesis report (samples) across Swedish universities.

Samples from ten universities across Sweden are collected. Most recent documents were given more importance.

8.2 Data Analysis:

Analysis of data is a process of inspecting, cleaning, transforming, and modeling data with the goal of highlighting useful information, suggesting conclusions, and supporting decision making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.

The process of data analysis consists of three stages:

(1) Initial Exploration

(2) Model building or pattern identification with verification/validation.

(3) Deployment (i.e.) the application of the model to new data in order to generate predictions.

Stage 1: Exploration.

This stage usually starts with data preparation which may involve cleaning data, data transformations, selecting subsets of records in case of data sets with large number of variables (fields) performing some preliminary feature selection operations to bring the number of variables to a manageable range(depending on the statistical methods which are being considered).

In this paper, data exploration starts with collecting the thesis samples from the large relational database. The feature selection depends on collecting the most recent Master level Computer Science samples from various universities across Sweden.

Stage 2 : Model Building and Validation.

This stage involves considering various models and choosing the best one based on their predictive performance. This may sound like a simple operation , but in fact , it sometimes involves a very elaborate process.

There are variety of techniques developed to achieve that goal many of which are based on so called “Competitive evaluation of models”, that is applying different models to the same data set and then comparing their performance to choose the best.

Based on this stage, Ephorus anti plagiarism software which is based on text based detection technique is used to carried out the analysis of plagiarism extent among the Swedish Universities.

Stage 3: Deployment.

The final stage involves using the model selected as best in the previous stage and applying in to new data in order to generate predictions or estimations of the expected outcomes.

The data collected from the first stage and the detection model selected from the second stage are used to test the data to find the possible plagiarism extent.

Sl.No	**Name of the Universities**	**Number of Samples**
1	Linköping [41]	32
2	Dalarna [42]	29
3	Halmstad [43]	29
4	Skövde [44]	29
5	Umeå [45]	22
6	Uppsala [46]	30
7	BTH [47]	12
8	KTH [48]	4
9	Linnaeus [49]	21
10	Mälardalen [50]	12
	Total	220

Table 2 : Swedish Universities and Samples.

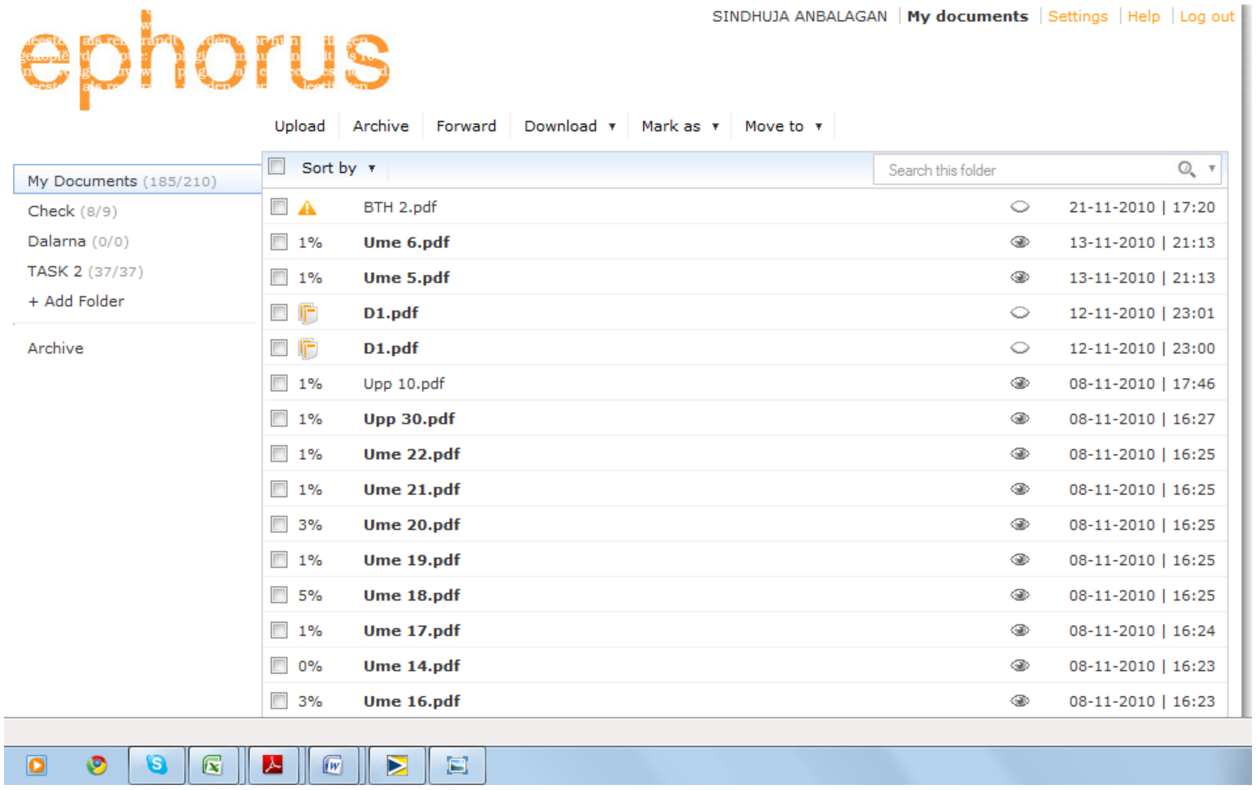
Selection of above samples :

The documents were selected from the respective university websites. Mode of selection is based on the recent Computer science Master's thesis report, for example papers belong to

2010, 2009, 2008 mostly, but in some cases recent papers were not available in the university websites, so the next available computer science Master's thesis reports were taken. The maximum possible samples from Master level computer science domain is taken across Swedish Universities and processed in the Ephorus anti-plagiarism software. For example, 32 samples were taken from Linkoping university, but only 4 documents were available in the kth university.

Step 2: Processing the documents in Ephorus software to find the possible sources of plagiarism.

The collected documents are then uploaded in Ephorus software, and the initial percentage of plagiarism was found.



⚠	BTH 2.pdf		21-11-2010 | 17:20
1%	Ume 6.pdf		13-11-2010 | 21:13
1%	Ume 5.pdf		13-11-2010 | 21:13
	D1.pdf		12-11-2010 | 23:01
	D1.pdf		12-11-2010 | 23:00
1%	Upp 10.pdf		08-11-2010 | 17:46
1%	Upp 30.pdf		08-11-2010 | 16:27
1%	Ume 22.pdf		08-11-2010 | 16:25
1%	Ume 21.pdf		08-11-2010 | 16:25
3%	Ume 20.pdf		08-11-2010 | 16:25
1%	Ume 19.pdf		08-11-2010 | 16:25
5%	Ume 18.pdf		08-11-2010 | 16:25
1%	Ume 17.pdf		08-11-2010 | 16:24
0%	Ume 14.pdf		08-11-2010 | 16:23
3%	Ume 16.pdf		08-11-2010 | 16:23

Figure 4: Ephorus homepage with Swedish university documents

Step 3:

In order to avoid the false positives, human intelligence is used to find out the final percentage of plagiarism. Each Ephorus document results is analyzed by human in order to remove the false positives, this will greatly helpful in finding the exact percentage of plagiarism. The final plagiarism percentage for each sample is given as follows.

Linköping	% Of P	Dalarna	% of P	Halmstad	% of P	Skövde	% of P	Umeå	% of P
Link 1	2	D 1	0	Ham1	6	Skv1	1	Ume1	1
Link 2	6	D 2	12	Ham 2	1	Skv2	4	Ume2	1
Link 3	4	D 3	29	Ham3	1	Skv3	1	Ume3	0
Link 4	3	D 4	1	Ham4	1	Skv4	1	Ume4	1
Link 5	3	D 5	1	Ham5	2	Skv5	1	Ume5	1
Link 6	1	D 6	3	Ham6	2	Skv6	1	Ume6	1
Link 7	3	D 7	1	Ham7	1	Skv7	3	Ume7	1
Link 8	2	D8	3	Ham8	2	Skv8	3	Ume8	1
Link 9	2	D9	1	Ham9	3	Skv9	1	Ume9	0
Link 10	1	D10	1	Ham10	1	Skv10	0	Ume10	2
Link 11	0	D11	3	Ham11	1	Skv11	2	Ume11	1
Link 12	5	D12	1	Ham12	2	Skv12	1	Ume12	3
Link 13	9	D13	0	Ham13	1	Skv13	1	Ume13	4
Link 14	2	D14	1	Ham14	1	Skv14	1	Ume14	0
Link 15	4	D15	1	Ham15	1	Skv15	1	Ume15	1
Link 16	4	D16	1	Ham16	1	Skv16	3	Ume16	3
Link 17	2	D17	1	Ham17	1	Skv17	2	Ume17	1

Link 18	1	D18	0	Ham18	1	Skv18	0	Ume18	5
Link 19	5	D19	1	Ham19	1	Skv19	3	Ume19	1
Link 20	4	D20	3	Ham20	1	Skv20	1	Ume20	3
Link 21	4	D21	3	Ham21	2	Skv21	1	Ume21	1
Link 22	2	D22	0	Ham22	1	Skv22	0	Ume22	1
Link 23	2	D23	7	Ham23	2	Skv23	2		
Link 24	4	D24	3	Ham24	8	Skv24	2		
Link 25	15	D25	1	Ham25	1	Skv25	2		
Link 26	4	D26	7	Ham26	1	Skv26	1		
Link 27	2	D27	8	Ham27	2	Skv27	0		
Link 28	3	D28	2	Ham28	8	Skv28	1		
Link 29	1	D29	7	Ham29	1	Skv29	1		
Link 30	17								
Link 31	4								
Link 32	1								

Table 3(i): Swedish Universities and Percentage of Plagiarism.

Uppasala	% of P	BTH	% of P	KTH	% of P	Linnaeus	% of P	Malardalen	% of P
Upp1	0	BTH1	1	KTH1	3	Line1	3	Malar1	3
Upp2	0	BTH2	9	KTH2	1	Line2	1	Malar2	1
Upp3	1	BTH3	4	KTH3	2	Line3	0	Malar3	1
Upp4	0	BTH4	1	KTH4	2	Line4	1	Malar4	3
Upp5	0	BTH5	1			Line5	5	Malar5	1
Upp6	0	BTH6	4			Line6	2	Malar6	2
Upp7	2	BTH7	1			Line7	1	Malar7	2
Upp8	2	BTH8	4			Line8	1	Malar8	3
Upp9	10	BTH9	2			Line9	1	Malar9	0

Upp10	1	BTH10	1			Line10	2	Malar10	1
Upp11	1	BTH11	1			Line11	1	Malar11	1
Upp12	1	BTH12	2			Line12	1	Malar12	7
Upp13	1					Line13	10		
Upp14	2					Line14	1		
Upp15	2					Line15	2		
Upp16	2					Line16	1		
Upp17	2					Line17	2		
Upp18	2					Line18	1		
Upp19	0					Line19	11		
Upp20	8					Line20	3		
Upp21	3					Line21	2		
Upp22	3								
Upp23	2								
Upp24	1								
Upp25	2								
Upp26	2								
Upp27	2								
Upp28	2								
Upp29	2								
Upp30	1								

Table 3(ii):Swedish Universities and Percentage of Plagiarism.

From the above table, only 5 documents out of total 220 (2 from Linkoping , 2 from Dalarna and 1 from Uppasala) has plagiarism percentage more than 10%. The manual anlaysis of these documents shows that they were plagiarized against the internet sources and not with the fellow students.

8.3 Threshold Setting

Example for threshold setting in the Academic Plagiarism detection:

The following is an example for determining the quality of thesis work , based on the final percentage of plagiarism. The threshold is user defined.

Threshold 1 T1 = <5 % - Ok.

Threshold 2 T2 = 5 to 20 % - Questionable.

Threshold 3 T3 = > 20% - Reject.

Based on this threshold levels, it is possible to assess the quality of the document.

From the above samples,

UNIVERSITY	**COUNT**		
	T 1 = < 5 % OK	**T2** = 5 – 20 % QUESTIONABLE	**T 3** = > 20 % REJECT
Linköping	26	6	-
Dalarna	23	5	1
Halmstad	26	3	-
Skövde	29	-	-
Umeå	21	1	-
Uppsala	28	2	-
BTH	11	1	-

KTH	4	-	-
Linnaeus	18	3	-
Mälardalen	11	1	-

Table 4: Threshold and Counts.

Step 4: Once the percentage of plagiarism has been found, STATISTICAL

DATA ANALYSIS is carried out using Minitab software.

8.4 Minitab software:

Minitab software is used for statistical analysis [51].

8.5 Working on Minitab :

The percentage of Plagiarism for each universities(data values) is uploaded in the software.

The mean , standard deviation, variance, minimum percentage value of plagiarism, maximum percentage value of plagiarism and range is calculated for each university.

8.6 Software Sensitivity Test:

The following tests were carried out in order to find the sensitivity of Ephorus software.

Test 1:

Processing my own thesis documents with references unremoved.

Test Result : Percentage of Plagiarism : 3%

Test 2:

Processing my own thesis document with references removed.

Test Result : Percentage of Plagiarism : 16%

Test 3:

Processing a document from internet without references.

Test Result : Percentage of Plagiarism : 100%

Test 4:

Processing the same document from internet with a change in synonyms.

Test Result : Percentage of Plagiarism : 100 %

Inference about the sensitivity test:

Test 1 result shows an example for accidental cloning and failure to cite the references. Test 2 result shows an example for person’s true work in their thesis, in this case 16% is used or referred from other’s work. Test 3 shows the performance and efficiency of software when a document is stolen from internet and used as if one’s own, here it detects completely 100%, which shows high efficiency of Ephorous anti plagiarism software. Test 4 , checks for the consistency in efficiency, when there is a change in the synonyms , which again proves the efficiency (100%)of Ephorus anti plagiarism software. From the test 4, it could be said that, Ephorus is not only based on detecting the clones, but it also has some kind of intelligence techniques which could find the duplicated clone with change in synonyms.

9.Results

9.1 Results:

After uploading the data values (i.e percentage of plagiarism for each university)

The following calculations are carried out [51].

Variable	Mean	St Dev	Variance	Minimum	Maximum	Range
Linköping	3.813	3.677	13.448	0.000	17.000	17.000
Dalarna	3.52	5.69	32.40	0.000	29.00	29.00
Hamstard	1.966	1.955	3.820	1.000	8.00	7.000
Skövde	1.414	1.018	1.037	0.000	4.000	4.000
Umeå	1.500	1.300	1.690	0.000	5.000	5.000
Uppsala	1.900	2.139	4.576	0.000	10.000	10.000
Bth	2.583	2.392	5.720	1.000	9.000	8.000
Kth	2.000	0.816	0.667	1.000	3.000	2.0
Linnaeus	2.476	2.874	8.262	0.000	11.000	11.000
Mälardalen	2.000	1.809	3.273	0.000	7.000	7.000

Table 5: Statistical Output.

9.2 Graphical Representation :

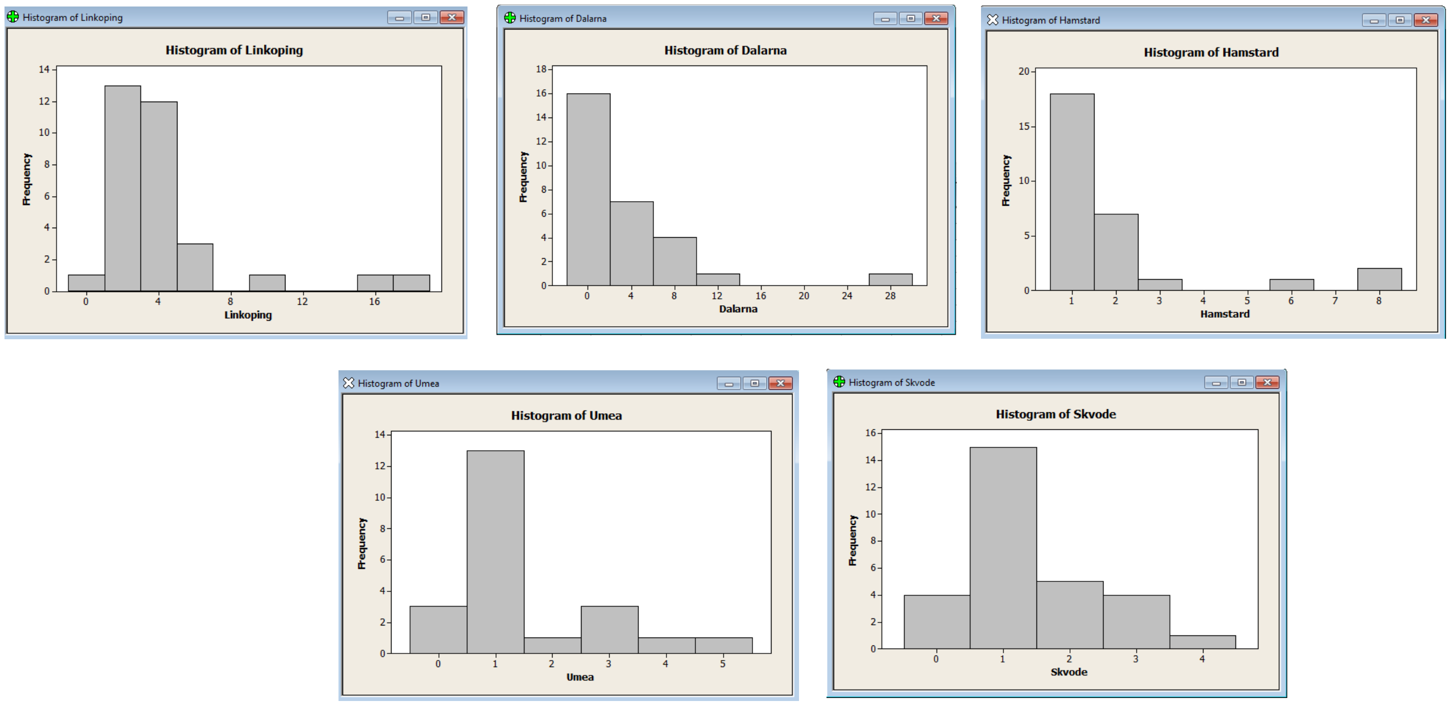


Fig:5(i) Graphical Representations

Scale:

X-axis : Percentage of Plagarism

Y-axis : Number of Plagarised Documents

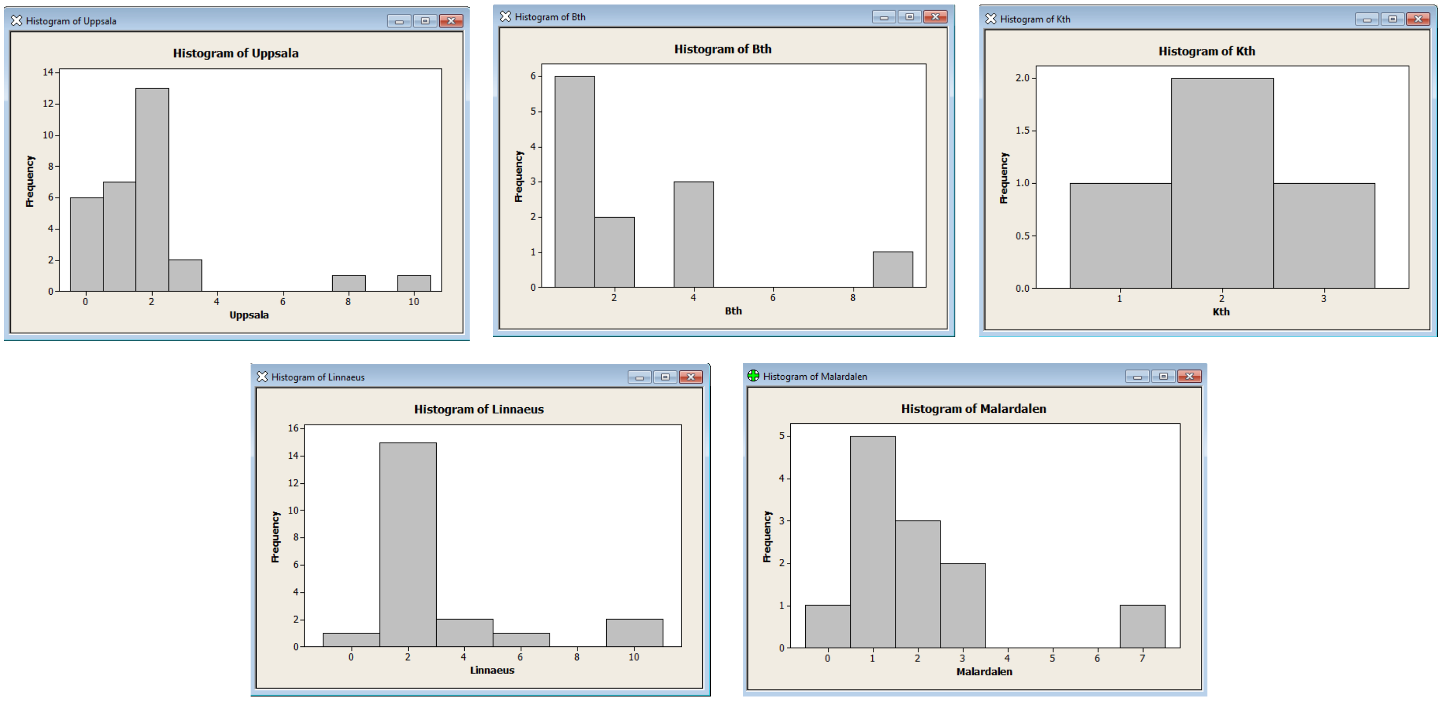


Fig:5(ii) Graphical Representations

Scale:

X-axis : Percentage of Plagarism

Y-axis : Number of Plagarised Documents

9.3 Conclusion:

The first part of the paper gives the detailed idea about the software clones. The choice among the six clone detection techniques largely depend on the problem domain.The Academic text based plagiarism detection is based on various factors like scope of search, analysis time, document capacity/batch processing, check intensity , precision and recall, so among various detection techniques Text based detection technique is highly efficient for plagiarism detection among the thesis report.

The later part of the paper , detects the plagiarism extent among Swedish Universities using Ephorus software which depends mainly on text based and some hybrid based detection techniques. The preventive measures are helpful in avoiding the clones. Exact refactoring is used to remove the clones. Simultaneous Editing is used to manage the clones.

From the above results, the overall percentage of plagiarism occurrence among Swedish universities is **2.3%(SE of the mean 0.253, confidence interval (1.8,2.8))** Out of the 220 thesis documents only 5 documents were above or equal to 10% of plagiarized content, and the manual analysis shows that they were plagiarized against the internet sources and not with the fellow students . The threshold setting shows that majority of documents(197 samples) falls under T1 class, very few documents(22 samples) falls under T2 class and only one document falls under T3 class. This could be either intentional or accidental. The main reason could be lack of citing the references.

The result gives that the plagiarism occurrence among Swedish universities is very low, which shows that Plagiarism is not a threat to Sweden’s education in Master level Computer Science department. Moreover, Software sensitivity analysis shows the efficiency of the Ephorus software plagiarism detection, and the important point to be noted is that Ephorus is not just doing clone detection it also has an intelligence technique that could find the change in synonyms, which is an added advantage. So Ephorus should be based on Jplag and other intelligence techniques for detecting the clones.

The study of software clones , clone detection methodologies, working on Ephorus and Minitab were really interesting and I prefer that awareness on anti plagiarism should be encouraged among students in order to respect other’s work and to discover new innovative ideas.

10.Discussion and Future Work

The use of DUP tool, which has a high efficiency in memory handling and recall precision will improve the results of the software plagiarism detection. Moreover the DUP clone detector appears to be more suitable for refactoring activities since it respects both syntactic integrity and include context dependencies in its clone. Its better to explore additional anti software plagiarism detection softwares like Turnitin and Urkund .

This paper is based only on Computer Science domain, when we extend the plagiarism check to other departments as well, we can get the exact figure of plagiarism occurrence among Swedish Universities..

Moreover, the figure 2.3% should be compared to other Computer science departments across the world to have a clear insight about the research.

11. References

11.1 References:

[1] B.Baker on finding duplication and near- duplication in large software systems , In WRCE , pp 86-95,1995.

[2] C.K.Roy and J.R.Cordy. An Empirical study of function clones in open source software. In WRCE 2008, pp 81-90, 2008.

[3]M.Kim and G.Murphy. An Empirical study of code clones in Genealogies. In FSE,pp 187-196,2005.

[4] L.Aversano , L.Cerulo and Massimiliano Di Penta. How clones are maintained : An Empirical study. In CSMR, pp 81-90,2001.

[5] C.Kapser and M.Godfrey. "Cloning considered harmful" Considered harmful. In WCRE,pp 19-28,2006.

[6] J.Haward Johnson. Identifying redundancy in source code using fingerprints.In proceeding of the 1993 conference of the centre for advanced studies conference (cascon' 93) pp 171-183, Toronto, Canada, October 1993.

[7] A.Chou, J.Yang, B.Cherf, S.Hallem and D.R.Engler. An Empirical study of Operating system errors. In proceeding of the 18th ACM symposium on Operating system principles (SOSP' 01). Pp 7388, Banff, Alberta, Canada, October 2001.

[8] M.Fowler. Refactoring : Improving the design of Existing code, Addison –Wesley, 2000.

[9] Bruno Lague, Daniel Proulx, Jean Mayrand, Ettore M.Merlo and John Hudeponl. Assessing the benefits of Incorportating function clone detection in a Development process. In proceeding of the 13th International conference on software maintenance (ICSM' 97), pp 314-321, Bari, Italy, October 1997.

[10] Ira Baxter, Andrew Yahin, Leonardo Moura, Marceto Sant Anna, ssClone detection using Abstract syntax trees.In proceedings of the 14th International conference on software maintenance (ICSM' 98), pp 368-377 , Bethesda, Maryland, November 1998.

[11] Mirybang Kim, Gail Murphy. An Empirical study of code clone Genealogies. In proceedings of the 10th European software engineering conference held jointly with 18th ACM

SINSOFT International symposium on foundation of software engineering(ESEC/SINGSOFT CSE 2005' 05) pp 187-196, Lisbon, Portugal, September 2005.

[12] Mattias Rieger, Stephane Ducasse, Michele Lanza. Insights into system- wide code duplication. In proceedings of the 11th IEEE working conference on Reverse engineering (WCRE' 04), pp 100-109, Derf University of Technology. Netherland, November 2004.

[13] A Mutation/Injection – based Automatic framework for evaluating code clone detection tools – Chanchal K.Roy and James R.Cordy, School of Computing, Queen's University Kingston, ON, Canada K7L 3N6 {cordy,cordy}@cs.queensu.ca

[14] Cory Kapser and Michael W.Godfrey "clones considered harmful". In proceedings of the 13th working conference on Reverse engineering(WCRE ' 06) pp 19-28, Benevento, Italy, October 2006.

[15] James.J. Hunt and Walter .F.Tichy. Extensible language aware merging. In proceedings of the International conference on software maintenance (ICSM' 02), pp 511-520, Montreal, Canada, October 2002.

[16] Mathias Rieger. Effective clone detection without language barriers Ph.D thesis, University of Bern, Switzerland, June 2005.

[17] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CC Finder : A Multilinguistic token based on code clone detection system for large scale source code. Transactions on software engineering Vol. 28(7): 654-670, July 2002.

[18] Brenda Baker. On finding duplication and near duplication in large software systems. In proceedings of the second working conference on Reverse engineering (WCRE ' 95), pp 86-95, Toronto, Ontorio, Canada. July 1995.

[19] W.K.Chen, B.Li and R.Gupta. Code compaction of matching single-entry multiple exit regions. In proceedings of the 10th Annual International static analysis symposium(SAS ' 03), pp 401-417, San Diego, USA, June 2003.

[20] Chanchal kumar Roy and James R.Cordy . A Survey on software clone detection research ,Queen's University at Kingston, Ontario, Canada, 2007.

[21] Brenda S. Baker. A Program for Identifying Duplicated Code. In Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.

[22] Raghavan Komondoor and Susan Horwitz. Using Slicing to Identify Duplication in Source Code. In Proceedings of the 8th International Symposium on Static Analysis (SAS'01), Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.

[23] Brenda S. Baker. Parameterized diff. In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99), pp. 854-855, Baltimore, Maryland, USA, January 1999.

[24] Wuu Yang. Identifying syntactic differences between two programs. In SoftwarePractice and Experience, 21(7):739755, July 1991.

[25] V. Wahler, D. Seipel, Jurgen Wolff von Gudenberg, and G. Fischer. Clone detection in source code by frequent itemset techniques. In Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM'04), pp. 128135, Chicago, IL, USA, September 2004.

[26] Williams Evans, and Christopher Fraser. Clone Detection via Structural Abstraction. In Proceedings of the 14th Conference on Reverse Engineering (WCRE'07), Vancouver, BC, Canada, October 2007(to appear, available as Technical Report since August 2005).

[27] Raghavan Komondoor. Automated Duplicated-Code Detection and Procedure Extraction.Ph.D. Thesis, 2003.

[28] Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309, Stuttgart, Germany, October 2001.

[29] Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic
Detection of Function Clones in a Software System Using Metrics. In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96), pp. 244-253, Monterey, CA, USA, November 1996.

[30] K. Kontogiannis, M. Galler, and R. DeMori. Detecting code similarity using patterns. In Working Notes of 3rd Workshop on AI and Software Engineering, 6pp., Montreal, Canada, August 1995.

[31] Filippo Lanubile, and Teresa Mallardo. Finding Function Clones in Web Applications. In Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR'03), pp. 379-386, Benevento, Italy, March 2003.

[32] Rainer Koschke, J.-F. Girard, M. Wrthner. An Intermediate Representation for Reverse Engineering Analyzes. In Proceedings of the 5th Working Conference on Reverse Engineering (WCRE'98), pp. 241-250, Honolulu, Hawai, USA, October 1998.

[33] Robert Tairas, Jeff Gray. Phoenix-Based Clone Detection Using Suffix Trees. In Proceedings of the 44th annual Southeast regional conference (ACM-SE'06), pp. 679- 684, Melbourne, Florida, USA, March 2006.

[34] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, Kostas Kontogiannis. Measuring Clone Based Reengineering Opportunities. In Proceedings of the 6th International Software Metrics Symposium (METRICS'99), pp. 292-303, Boca Raton, Florida, USA, November 1999.

[35] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini:Maintenance support environment based on code clone analysis. In Proceedings of the 8th IEEE Symposium on Software Metrics (METRICS'02), pp. 6776, Ottawa, Canada, June 2002.

[36] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini: Maintenance support environment based on code clone analysis. In Proceedings of the 8th IEEE Symposium on Software Metrics (METRICS'02), pp. 6776, Ottawa, Canada, June 2002.

[37] http://www.suite101.com/content/a-definition-for-plagiarism-a10232

[38] http://www.ephorus.com/home

[39] http://www.ephorus.com/media/88699/it's%20learning%20manual.pdf

[40]http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm

[41]http://liu.divaPortal.org/smash/searchlist.jsf?searchtype=simple&freetext=Computer%20science%20thesis%20report

[42] http://dalea.du.se/theses/default.aspx

[43] http://hh.diva-portal.org/smash/searchlist.jsf?searchId=1

[44] http://his.diva-portal.org/smash/searchlist.jsf?searchId=2

[45] http://umu.diva-portal.org/smash/searchlist.jsf?searchId=1

[46] http://uu.diva-portal.org/smash/searchlist.jsf?searchId=2

[47] http://www.bth.se/fou/cuppsats.nsf/$$Search

[48] http://kth.diva-portal.org/smash/searchlist.jsf?searchId=1

[49] http://lnu.diva-portal.org/smash/searchlist.jsf?searchId=1

[50] http://mdh.diva-portal.org/smash/searchlist.jsf?searchId=1

[51] http://www.minitab.com/en-US/default.aspx