DESIGN AND ESTIMATION OF FEATURES BASED SOFTWARE BIRTHMARK

By

Shah Nazir

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Department of Computer Science,
University of Peshawar, Peshawar, Pakistan
September, 2015

DESIGN AND ESTIMATION OF FEATURES BASED SOFTWARE BIRTHMARK

By

Shah Nazir

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Department of Computer Science,
University of Peshawar, Peshawar, Pakistan
September, 2015

To my loving family

Certificate of Approval

It is certified that this thesis submitted by Mr. Shah Nazir, titled "Design and Estimation of Features Based Software Birthmark" is up to the requirements and sufficient for the award of the degree of Doctor of Philosophy in Computer Science. All the work done is solely the effort of the scholar and an adequate appreciation is given to the work of others which is mentioned as reference material.

Supervisor
Dr. Sara Shahzad
Assistant Professor
Department of Computer Science,
University of Peshawar, Peshawar, Pakistan

External Examiner
Prof. Dr. Zahid Hussain
Dean, Faculty of Science, Quaid-e-Awam University of Engineering,
Science & Technology, Nawabshah, Sindh, Pakistan

Chairman
Dr. Saeed Mahfooz
Associate Professor
Department of Computer Science,
University of Peshawar, Peshawar, Pakistan

Declaration

I, hereby declare that this research thesis submitted to the Department of Computer Science, University of Peshawar, Peshawar, Pakistan, is my own and original research.

Shah Nazir

Acknowledgement

I consider it my supreme responsibility to offer my humble obedience before Almighty Allah, Who enabled me to complete this research work. I feel great delight and happiness in expressing heartfelt gratitude to my research supervisor, Dr. Sara Shahzad, for her motivating and stirring guidance, devotion of time, valuable suggestions and chivalrous behavior in completing this research work.

I am thankful to the Dean Faculty of Numerical and Physical Sciences Prof. Dr. Mohammad Abid, Chairman Dr. Saeed Mahfooz and other teaching faculty of the department, whose dignified attitude with highly specialized guidance, skilled advice, encouragement and wisdom made the completion of this work possible.

I am also thankful to my brothers and sisters whose love, encouragement, and assistance incented me to strive towards my goal.

Last but not the least, I am thankful to my sweetest parents, who helped and encouraged me in every way from day one at school up till this PhD research, and embellished me with prayers.

Shah Nazir

Abstract

Software theft and piracy are rapidly increasing problems of modern day software industry. This involves copying and reusing software without proper authorization enforced by the software license agreements. Piracy of software ultimately results in big financial loss to the copyright holder. General user community is not well aware of this serious crime, and most of the time people think that it is not important for them to worry about. Different advanced techniques are being used by practitioners to detect and prevent software theft, including software watermarking and software fingerprints. But the use of countermeasures, such as code obfuscation and optimization of software for the semantic transformation of watermark, has made their use partially illogical. In the same direction, a lot of research has been conducted to develop the concept of software birthmark, which is now a widely accepted technique to detect software theft. A software birthmark is based on the inherent properties of software. Accordingly, researchers have proposed different categories and types of software birthmark based on some defined attributes. Two attributes, credibility and resilience, are considered as the most important attributes of a software birthmark. Although the concept and types of software birthmark is extensively studied by software developers and practitioners, there is still a lack of knowledge and understanding about how to estimate a birthmark to analyse the extent of piracy.

The aim of this research is to promote conscious efforts made during software development so as to incorporate well defined design features, resulting in software with strong birthmark, hence preventing software piracy. For this purpose, a feature-based software birthmark has been proposed which provides a closer and authentic

identification of software, and then ultimately be used for theft detection. This research also presents a formal estimation model for features-based birthmark which can be used to detect and investigate the extent of piracy in software. The model is tested through a case study. Results exhibit correctness and usefulness of the proposed features based software birthmark and its estimation model. A mathematical model is also presented, through which the birthmark of software(s) can be compared for analyzing extent of piracy.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

BSA Business Software Alliance

WPP Whole Program Path

CVFV Constant Value in Field Variable

SMC Sequence of Method Calls

IS Inheritance Structure

UC Used Classes

FL Fuzzy Logic

FIS Fuzzy Inference System

MF Membership Function

VL Very Low

L Low

M Medium

VH Very High

PCF Pre Conditional Features

IF Input Features

NFF Non Functional Features

FF Functional Features

PA Program Availability

Ru Runnable

IoC Identification of Components

PCnxt Program Context

PF Program Flow

PCnt Program Contents

PDS Internal Data Structure

PR Program Responses

CT Configurable Terminologies

CF Control Flow

NoSP Number of Statements in Program

Na Naming

F Functions

ID Interface Description

RLC Restrictions, Limitations and Constraints

SoP Size of Program

CD Comprehensive Documentation

GDS Global Data Structure

UI User Interface

IQ Internal Quality

A Automation

EoU Ease of Use

UF User Friendly

Sc Scalability

Ap Applicability

ICn Interface Connection

R Robustness

D Dependency

P Portability

S Scope

Std Standard

EQ External Quality

DCT Data and Control Transfer

FS Functional Specification

A Behavior

Fnl Functionality

IEEE Institute of Electrical and Electronic Engineers

ACM Association for Computing Machinery

Chapter 1

Introduction

Software birthmark is an inimitable quality of software used to detect software theft. Comparing birthmarks of software helps to identify pirated copies of the original software application. Software theft and piracy are two rapidly increasing problems which involve copying and reusing software without proper authorization, as enforced by software license and agreement. Piracy of software ultimately results in financial loss to the copyright holder. On the other hand pirates earn huge profits. General user community is not well aware of this serious crime, and most of the time they feel that it is not important for them to worry about. The following sections provide an overview of this serious problem and identify the goals and objectives of this research.

1.1 Piracy problem in software industry

Software industry has suffered huge financial loss due to piracy of software. Software piracy is performed by end-users as well as by the dealers. It causes serious problems which hinder in the success of the international software industry. It is a problem of illegal copying, installation, use, distribution or sale of software in any manner other than that is expressed in the license agreement. Pirates gain easy benefits from the sale of pirated software which ultimately affects the business of the software industry. The original licensed software offers a number of high valued benefits to the customers, including assurance of software quality, availability of upgrades, technical documentations, and lastly by less bandwidth consumption. On the other hand pirated

software does not provide such kind of facilities. If an organization is using pirated software, there might be risk of failure of the system, which might put the organization at risk of huge financial loss.

1.2 Identification of software theft

Software development industry has been employing different techniques for the detection and identification of software theft. These techniques mainly include advanced versions of software watermarking and fingerprints [1-14]. Software watermarks emphasize on the ownership of software programs by adding additional information to the software application. This additional information (that is, code) is the drawback of watermarking, as it takes extra space (for watermark) and may break the code in many cases. Fingerprint is used in tracking the intellectual property. Fingerprints embed a secret message to show the intellectual property and trace the original purchaser of the pirated software. Watermarking techniques have been used for some time as a remedy against software copy as well as for theft detection. Use of advanced techniques, such as code obfuscation, that is used for preventing malicious user to disclose properties of the original source program, and optimization of software used for semantic transformation of watermark, has partially made it illogical to use software watermarks. For this reason the concept of software birthmark has been developed and is now widely accepted as a technique to detect software theft. Software birthmark is a property based system which identifies the inherent characteristics of a program to check and show the originality of software. Most of the study on software birthmark focuses on how to describe appropriate properties of software which ultimately help to detect software theft.

Software features present all the information related to a software system. Software features collectively define and support a software system and its functionality. Some features are also the formal representation of a user centric organization of software or program functionality. These features are virtually interlinked performing different operations and due to these operations the software or program is considered to be a functional software system. The idea of identifying software features began from the source code theft detection [15]. One of the main problem with identifying software feature may be that the source code is not always available. Tamada et al. [16] considered java byte code set as a software features rather than source code. This is because a software feature is the well-known functional and non-functional formation of attributes and unique user-visible characteristics of software.

1.3 Background

Software watermark and birthmark are the most dominant methods used for the detection and identification of software theft and piracy. A birthmark is composed of some inherent characteristics of software which can be used for the theft detection. Watermark which is usually used in images, emphases the ownership of the software by embedding additional information (in the form of an image or text) which may be visible to the user. Whereas, a birthmark is an inherent characteristic that is derived from within the software. Software birthmarks have two most important properties that are credibility and resilience.

There are different categories of software features which can be defined for a software. A clear understanding of these features and their organization into logical categories helps

to understand the program code. This understanding is important to identify similarities among instance, or copies of presumably the same software application. A software program is a collection of different software features of certain types. The analysis of program code eventually helps in identifying similarities among more than one instance or copies of presumably same software application (that is, the program). The identification of similarities hence facilitates software piracy and theft detection.

There are different categories under which software features can be placed. For example, functional features, structural features, quality features, and so on. Y. Guo et al. [17] provide a categorization as input software features, self-software features and output software features. Silvio and Yang [18] categorized the features into syntactic and semantic features. Syntactic features deal with the structure of the program, while semantic features deal with the meaning of the program.

1.4 Motivation

Advancement in the field of software piracy is increasing day by day. Software piracy creates a serious problem for software industry. Researchers try to come across a methodology which can easily identify the piracy or theft of software. Yet, still, there is a lack of methodologies which can identify the software piracy in an efficient way.

Along with this the existing work related to software birthmark does not give much guidance to measure the extent of piracy by investigating a software birthmark.

To help software piracy detection process, it would be ideal to have a methodology which estimates a birthmark on the basis of any and predefined criteria according to the

requirements of the specified software. The aim of this research is to propose the process of estimation of software birthmark in terms of credibility and resilience, which have been accepted by the research community as the two most important properties of birthmark. Furthermore, this study also identifies other important attributes of software which may provide more important information about the software and further help in detecting software piracy. These attributes can be some inherent features of a software.

The purpose of using different features is to consider these features as a software birthmark. A collection of all the features of a program can help to provide a more close and authentic identification of a software program and then ultimately be used for theft detection. That feature based birthmark can then be estimated to understand the extent of piracy in software.

1.5 Research problem

Software piracy has turned out to be a major concern due to an extravagant development of software industry and the availability of software(s) on the Internet. Broad research into the techniques of software piracy detection has prompted the development of techniques like software watermarking, software finger prints, and lately the software birthmarks. A birthmark is based on intrinsic characteristic(s) of software which can be successfully used for software theft and piracy detection.

Different types of software birthmark(s) have been designed, dependent upon different programming languages and software design. Still, there is a lack of knowledge about how to estimate a birthmark to analyse the extent of piracy in software. There is a need to promote conscious efforts made during software development so as to incorporate well

defined design features, resulting in software with strong birthmark, hence preventing software piracy.

1.6 Goals and objectives

Following are the main goals and objectives achieved by this research.

- To estimate existing software birthmark(s) on the basis of credibility and resilience.
- To identify a plausible set of specific software features which provide a unique identity to a software as birthmark.
- To provide a method for estimating feature based software birthmark on the basis of credibility and resilience.
- To design a mathematical model for comparing features based software birthmark of software(s) to be analyzed for piracy detection.

1.7 Thesis outline

The chapter wise summary of the thesis is given below;

Chapter 1 provides an introduction to piracy problem in software industry, and software theft, research motivation, and goals and objectives.

Chapter 2 deal with the literature review of existing techniques for identification of software theft and piracy, types of software birthmark identification of software features and comparison of birthmark, watermark, fingerprints, clone and plagiarism techniques used for detection of piracy and theft.

Chapter 3 briefly discusses the details of proposed methodology that is rules based approach for estimating software birthmark. Use of estimation process, fuzzy logic, the algorithm for designing a rule based model, results, and discussion are also part of this chapter.

Chapter 4 is about the identification of software features. In this chapter four different categories of software features are identified. The categories of features include preconditioned software features, input features, nonfunctional features and functional features. These four categories of features are further subdivided into 36 different features.

Chapter 5 provides the details of estimation of software features based birthmark. Initially a set of features that are already identified in chapter no. 4 are used for the purpose of estimation of software birthmark. Fuzzy logic has been used as a tool for estimation of software features based birthmark. The results of the proposed methodology are evaluated and validated by the help of a case study.

Chapter 6 provides the details of the mathematical model designed to compare software(s) on the basis of features based software birthmark. These features are categorized in the form of differential system. Exact solution of these features has to be produced and then be compared with the solution of duplicate copy of the software.

Chapter 7 provides the conclusion and future work of the proposed research work regarding design and estimation of feature based software birthmark.

Chapter 2

Literature review

According to Business Software Alliance (BSA) report [19], in 2013, 43 percent of the software installed around the world on the personal computers was not properly licensed. The commercial value of this unlicensed software is 62.7 billion dollars. Ginger Myles and Christian Collberg [20] define three main threats to software industry which include illegal reselling of the legal software program, software tampering and malicious reverse engineering.

2.1 Techniques for the identification of software piracy and theft

Several diverse approaches have been in use for the detection of software piracy. These techniques include software watermarking [1-3, 7, 9-14, 21, 22], finger prints [23, 24], and software birthmark [4, 5, 20, 25-34]. Software watermark is used to show the ownership of the program. It needs additional code or information embedded to the software or program for showing the ownership of the program. Software fingerprints are used to show the intellectual property of the software.

Software cloning is another similar technique used to identify similarities in the code. Copying of whole code or part of code and pasting it in another part of the code is called software clone [35]. Different techniques have been presented for identification of clone [36]. A complete detail of software clone detection can be found in [35]. Plagiarism detection is also a very similar area to software birthmark that is used for detecting the theft of source code and finding similarity between the original and decompiled source

code. Some of the techniques used for plagiarism detection are named Moss and Winnowing etc. [15, 37-41].

Apart from these techniques, software birthmark is the inherit characteristic of a software or program used for the detection or checking of originality in software, and to show wither the software or program is a copy of another or not. Birthmark is important for identification and detection of software piracy, as it cannot be destroyed.

2.2 Software birthmark and its properties

Software birthmark is inherent characteristics of software which can be used for different purposes but used for the identification and most important one is the detection of software theft and piracy. In the literature available till now, researchers have considered two important properties of software birthmark which are used to evaluate their effectiveness, these are credibility and resilience. But Y. Zeng et al. [42] reports that not many theoretical frameworks are available that properly analyze and verify the success of software birthmark. The evaluation of software birthmark is mainly done through experiment. They have presented a semantic based abstract interpretation framework. This model is described over credibility and resilience. With the help of static n-gram birthmark and static API birthmark the effectiveness of the framework is verified. G. Myles and C. Collberg [20] presented a technique called “Whole Program Path Birthmarking” for detecting the software theft. Their technique is based on complete control flow of the software program. They used credibility and resilience to evaluate the efficiency of the technique. The technique demonstrates that the whole program path birthmark is more resilient than other birthmark techniques. Furthermore, the technique

also showed that even if an embedded watermark is destroyed by program transformation the birthmark can still identify the theft.

2.3 Static and dynamic software birthmark

H. Tamada et al. [43] proposed dynamic software birthmark. This birthmark can be extracted when windows applications are under execution. Z. Xin et al. [44] pointed out the weaknesses of existing techniques on software birthmark and designed semantic safe system call replacements for taking in the birthmark efficiently although if the performance overhead is low. P. P. F. Chan et al. [29] proposed a dynamic software birthmark system based on object reference graph for systems designed in java. The method was evaluated for huge programs. The results showed that the method was useful in detecting the code theft. K. Fukuda and H. Tamada [45] proposed a dynamic birthmark for Java Virtual Machine that is based on operand stack runtime behaviour. Y. Bai et al. [46] presented a dynamic K- gram based software birthmark for identification of origin of program. H. i. Lim et al. [47] proposed a static birthmark based on control flow edge for java programs. They evaluated their birthmark on credibility and resilience. H. Park et al. [5] used static API trace birthmark for the detection of Java based programs theft. Their method also evaluates the birthmark in term of credibility and resilience. The experimental result of the method shows that static API birthmark can detect similar modules of two packages whereas other birthmark techniques to do so fail. X. Xie et al. [48] proposed a static birthmark for the k-gram and their weights. The weight is computed by analyzing rate of change in k-gram frequency of the original and transformed program.

Y. Mahmood et al. [49] proposed a software birthmark technique named as method based similarity level. Through this method the code elements and their properties can be found. The method also detects changes occur in the program. Y. Wang et al. [50] proposed the operand stack dependence based static software birthmark for the problem of semantic lost when extracting birthmark with the help of k-gram algorithm.

X. Zhou et al. [51] presented a birthmarking technique for the identification of program that is based on the static and dynamic component dependence graph. The two important properties that are credibility and resilience of birthmark are compared with the Whole Program Path (WPP) birthmark and through the results it is shown that their technique is more efficient than the WPP birthmark. Guang Sun [52] extended the idea of birthmark based on component dependence graph with clustering. Their results show that the proposed birthmark is more stable than the WPP and component dependence graph. J. Choi et al. [53] proposed a static birthmark scheme for the identification of Windows executable files using the import address table. L. Ma et al. [54] presented a static software birthmark for the detection of software piracy and similarity calculation. The birthmark is composed of instruction words and their frequencies. The instruction word having semantics of program while instruction word frequency shows the difference between implementation details of programs. H. Kim et al. [55] presented a polymorphic attack against sequence based software birthmark. D. Lee et al. [56] proposed a method in which birthmark can be extracted through instruction categorization that satisfy resilience and uniqueness.

2.4 Birthmark for detection and identification of software theft

Watermarking techniques have been used as a remedy against software copy as well as for the theft detection. With the development of counter-techniques like watermark removal and destruction the usefulness of watermarks has been compromised. Also the use of advanced techniques such as obfuscation and optimization for semantic transformation of watermark had completely made it illogical to use software watermarks for this purpose. The concept of software birthmark was then developed as a technique to detect the software theft.

Tamada et al. [16] proposed the first birthmark which consists of four different birthmarks, namely; constant value in field variable (CVFV), sequence of method calls (SMC), inheritance structure (IS), and used classes (UC). This birthmark technique has been successfully used by the industry for software theft detection. Also, Y. Zeng et al. [42] presented a semantic based abstract interpretation framework for software birthmark. G. Myles and C. Collberg [20] presented a technique of "Whole Program Path Birthmarking" that was based on complete control flow of the software program. They used credibility and resilience to evaluate the efficiency of the technique and the technique also demonstrates that the whole program path birthmark is more resilient than other birthmark techniques.

T. Kakimoto et al. [34] analyzed the birthmark similarity in ArgoUML and visualized them using multi-dimensional scaling. Y. Wang et al. [57] used CHI (χ^2 statistics) for the characteristics selection in text classification and bring in an instruction words software birthmark selection. The algorithm makes sample program for protected program and take out instruction word from sample program according to instruction word library. To

find out their correlation the χ^2 statistics is calculated for each instruction word in and program. The experimental results of the algorithm show that the selection algorithm has much enhanced the robustness and credibility of the birthmark. S. Choi et al. [31] proposed a static API software birthmark for Windows binary executable. They compared 49 Windows executable and showed that their birthmark can differentiate and detect the copies. The birthmark is compared with the Windows dynamic birthmark and showed that it is more suitable for GUI application. H. Lim [26] presented a customized method of k-gram birthmark which permit the small changes of programs by applying partial matching of k-gram. The experimental result shows that customizing the k-gram birthmark improves the properties of birthmark that are credibility and resilience.

2.5 Identification of software features

The purpose of identifying different features is to consider them as software birthmark. Software can be dissected in multiple categories of different software elements which can be termed as features of particular software. K. C. Kang et al. [58] presented the concept of feature oriented domain analysis (FODA). The purpose of this study was to perform domain analysis and explain the products of the domain analysis. J. Kalaoja [59] emphasised on the feature modelling of embedded software systems. Several studies exists on identifying different software features to define birthmark, such as feature selection model for software defect prediction [60], identification of steganography software based on feature matching [61], and identification of software theft based on multi attributes features [17]. A collection of all the features of a program may help to

provide a more close and authentic identification of a software program and then ultimately be used for theft detection.

2.6 Comparison of software birthmark, watermark, clone, fingerprints and plagiarism detection

Table 2.1 comparison of software birthmark, watermark, clone, fingerprints and plagiarism

	Method	**Advantages**	**Drawback**
Watermark	Use additional information for showing the ownership of program Such as; Robust object watermarking [13], method for watermarking Java object [12], dynamic path based software watermarking [10], A Chaos-Based Robust Software Watermarking [3], Tamperproofing a software watermark by encoding constants [62], abstract interpretation	Strong evidence for showing the ownership of the program, as the program is encoded by some ownership symbol (watermark in the form of image or text)	Additional information can be erased (destroyed) through advance techniques, such as code obfuscation or optimization. Take additional storage space in memory (for watermark)

	based semantic framework for software watermark [63] etc.		
Fingerprints	Use of digital signature for showing the intellectual property of program Such as; Dynamic graph based software fingerprinting [24], winnowing [39], fingerprint for copyright software protection [23] etc.	Everyone can verify the copyright ownership of the software that is fingerprinted	Digital signature can be erased through cryptographic technique
Clone	Finds similarity in code by finding duplicate redundant code Such as; Clone detection using abstract syntax trees [36], etc	Finds similarity in code by finding duplicate code	Copy and paste
	Compute similarity at the source code level by	Show similarity in source text	Most of the time source code is not

Plagiarism Detection	comparing the source code with the duplicate code, such as; Moss [15], detection of similarity in student programs [38], DKISB [41], etc.	Similarity is source code can easily be find	available
Birthmark	Uses the inherent characteristics of program (function calls, method structure, etc) to show the originality of program. Such as; Java byte code based birthmarks, K-gram Instruction words based software birthmark etc. [4, 5, 26, 28, 29, 31, 32, 34, 41-43, 45-48, 53, 54, 56, 64-66]	Can be used when there is limitation of storage space. The other techniques fails to detect piracy, while birthmark detect the piracy, as it works on the basis of inherent characteristics of a program	Technical complexity. Does not show who is the owner of the program

Summary

This chapter presents literature review related to software industry problems to software piracy. The techniques for software identification and detection purpose of software theft or piracy, software birthmark and its properties, static and dynamic birthmark, birthmark for the detection and identification of software theft and identification of software features are also presented in this chapter. A comparison to software watermark, fingerprints, clone detection, plagiarism detection and birthmark is given at the end of this chapter.

Chapter 3

Rules based approach for estimating software birthmark

Estimation of software birthmark(s) can play a key role in understanding the effectiveness of a birthmark. In this chapter a new technique is presented to evaluate and estimate software birthmark based on the two most sought after properties of birthmarks, which are credibility and resilience. For this purpose the concept of soft computing, such as probabilistic and fuzzy computing has been taken into account and fuzzy logic is used to estimate properties of software birthmark. The proposed fuzzy rule based technique is validated through a case study. The results gathered from the case study show that the proposed technique is successful to assess the specified properties of the birthmark. This, in turn, shows the amount of effort which will be required to detect the originality of the software based on its birthmark.

The following sections define the proposed methodology to estimate software birthmarks.

3.1 Software piracy

Software industry has faced huge financial losses due to the piracy of software. Software Piracy is performed by end-users as well as the dealers. Software piracy causes serious problems which hinder the success of the international software industry. Piracy of software is a global problem of illegal copying, installation, use, distribution or sale of software in any manner other than that is expressed in the appropriate license agreement. The pirates gain easy benefits from the sale of pirated software which ultimately affects

the business of the software industry. Figure 3.1 shows how software piracy occurs from its original business market.

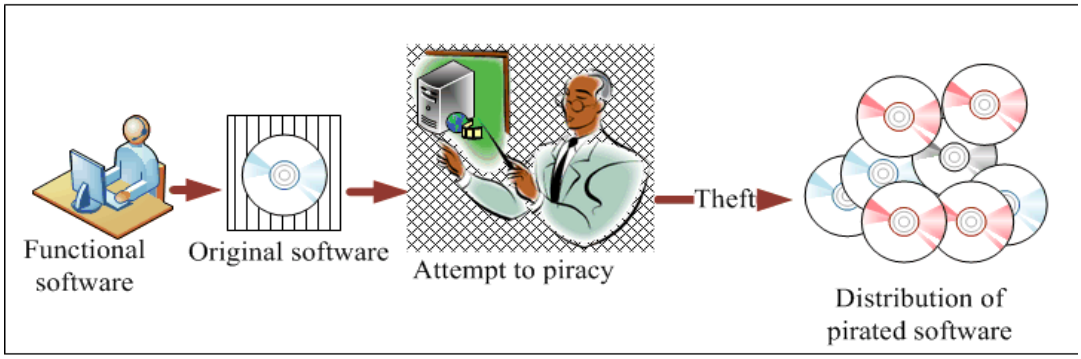

Figure 3.1. Software piracy

The original licensed software offers a number of high valued benefits to the customers, including assurance of software quality, availability of upgrades, technical and manual documentations, and less bandwidth consumption. On the other hand pirated software does not give such kind of facilities. There might be risk of failure of the system, if an organization is using pirated software, and it might put the organization at risk of huge financial loss.

3.2 Software birthmark and watermark

So far, different techniques are being used for theft detection of software such as [67] and [68]. Software birthmark is a promising technique used for the detection of software theft [42]. Birthmark does not embed additional code or information in any form to the original program. Software birthmarks only extract the inherent characteristics from the original program to detect the originality of program [20]. Software birthmark establishes an identity for software to detect if a program is a copy of any other program (partially or as a whole). It does not shows who the original owner of the program is or who is guilty

of software piracy [20]. While software watermarking asserts the ownership of the programs by adding extra information to the original program before it is publically available. Software watermarks identify software from the embedded information/code. Both the techniques can be combined to provide a stronger verification mechanism to detect theft. Birthmark can be used where there is a limitation of storage space as watermarking uses additional storage space. Also, in many situations watermarks fail, for example, if an attacker is able to apply obfuscation that destroys watermarks. In such situations software birthmarks provide evidence of piracy or software theft [20].

3.3 Properties of software birthmark

In order to estimate the success of software birthmarks, researchers typically consider two properties, which are credibility and resilience [68]. Credibility requires that the birthmark of the two programs must be different; whereas the resilience states that the birthmark should be preserved and not destroyed in any circumstances.

According to Tamada et al. [69] software birthmark satisfy the following two important properties which indicates that the two independently implemented programs should be different.

Property 1. Let P, Q be two independently written programs which achieve the same task, then f is credible if $f(P) \neq f(Q)$.

Property 2. Let P` be the program obtained from P by applying semantic preserving transformation T. f is resilient to T if $f(P) = f(P`)$.

Property 1 indicates that the birthmarks falsely showing that Q is a copy of P. This situation will occur with the separately implemented programs that achieve the same task.

Property 2 relates to identifying a copy in the occurrence of transformation. It is wished that a birthmark could be used to detect a copy if some transformation has been applied to the program.

In the existing literature on software birthmarks, there is lack of a formal model which closely estimates the birthmark of software based on the properties of credibility and resilience. The proposed methodology helps to estimate the birthmarks of software based on these properties.

3.4 Estimation

In the context of software birthmark, estimation means to check the extent of software piracy (exact copy or partially pirated). An accurate estimate of software birthmark with can help to identify the extent of piracy and theft. General review and static analysis of software cannot provide much information which is required to figure out the extent of piracy in software. Besides this, there are many other issues that also arise due to different perspectives of software design (for example, code complexity, vagueness etc.). Software birthmark estimation which is based on the important properties of birthmark that is credibility and resilience will easily identify the level to which piracy of the software belongs.

3.5 Use of estimation of software birthmark

Different techniques are already in use for the protection of software from attack such as [23, 67, 70-73]. Estimation is also being performed in some cases. Software watermark is already estimated by some researchers [74-77]. But birthmark have not been estimated

yet, which is also necessary for detecting the originality of software and to show that wither a program is a copy of another program or not. If there is a methodology that can estimate the birthmark of the software, then one can easily judge the success of birthmark in term of detecting software piracy. This estimation will enhance business of software industry and will grow up its economy in the market.

3.6 Fuzzy logic

Fuzzy logic concept was developed by A. Z. Lofti in 1965 [78]. It is a mathematical concept which deals with managing uncertain and vague information. Fuzzy logic is also used as systems control and analysis design model. It minimizes the time for engineering development and for extremely multifaceted schemes. It helps in providing solution for the problems which are complex to model [79, 80]. Fuzzy set theory, defined over the concept of fuzzy logic, has been successfully used for solving diverse problems in different fields of daily life. Fuzzy set is the extended form of traditional sets and is extremely beneficial for decision making in uncertain and vague situations. It facilitates a formal procedure to arrange vague information in such a way that it can be used for making decisions. A fuzzy set is based on some membership functions (MF) which represent the degree of an element, and the ratio of its value is between 0 and 1 [81]. The elements can be plotted as; element "x" belong to M, $\mu_M(x) = 1,\ \&\ if\ not\ \mu_M(x) = 0$.

Details of fuzzy logic concept have been defined by Zadeh [78], however the major parts of the fuzzy system are; fuzzy inference system, known as “fuzzification”, which transforms discrete classification inputs to continuous classification input. On the basis of

inference engine, FIS processes the rules in fuzzy domain and finally "de-fuzzify" it to real world values [82]. Figure 3.2 shows the process of the fuzzy model.

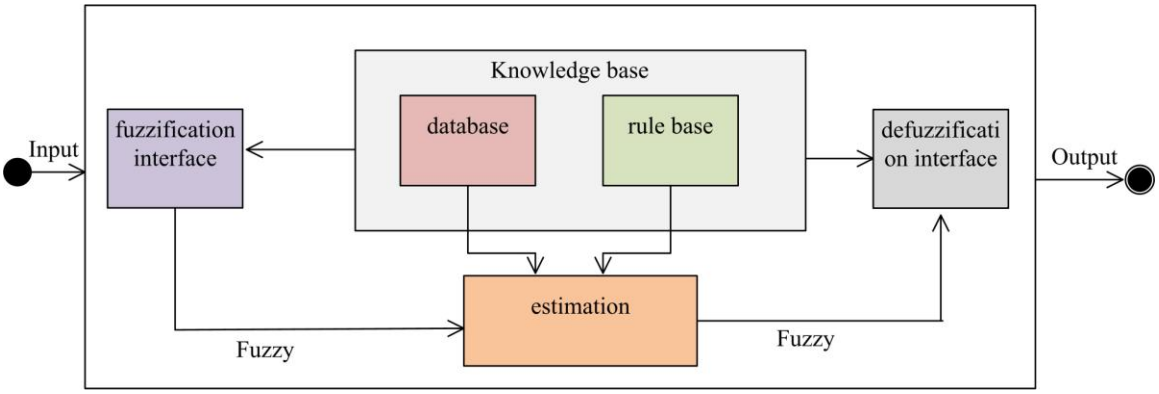

Figure 3.2. Process of fuzzy model

For the implementation of the proposed method for estimating software birthmarks, Fuzzy logic tool of Matlab is used [79]. Following are the details of the implementation (along with snapshots from the Fuzzy logic tool) regarding the estimation process.

In the proposed method the membership functions named as **mf1** is in the range of (0-19), **mf2** in range (20-39), **mf3** in range (40-59), **mf4** in range (60-79) and **mf5** in range (80-100) are defined. Also, to plot fuzziness triangular membership functions are defined and used to represent weights. Each triangular membership function has three parameters (l, m, u), which are defined as $l \leq m \leq u$. Figure 3.3 show the defined membership function for the proposed method.

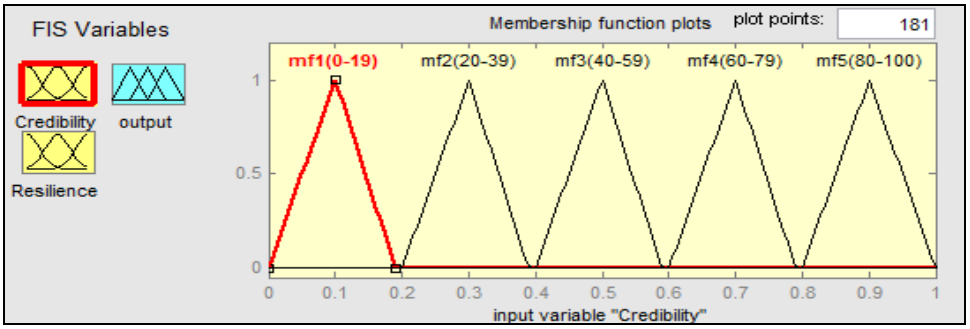

Figure 3.3. Membership functions for input property "credibility"

Similarly, the same membership functions are designed for the “resilience” property.

3.6.1 Fuzzy Inference System

FIS is a system in which rules can be planned for user specific purpose (estimation). These rules are based on membership function(s) connected using logical operations. Logical operations are "IF THEN" rules. FIS can also be applied to different fields such as control system, data organization, skilled system, computer visualization and many more [79]. Two types of approaches are used for FIS, which are Mamdani and Sugeno [79]. On the basis of fuzzy inference system different rules can be processed and will provide results accordingly. Figure 3.4 show the fuzzy inference system for our proposed model of estimation.

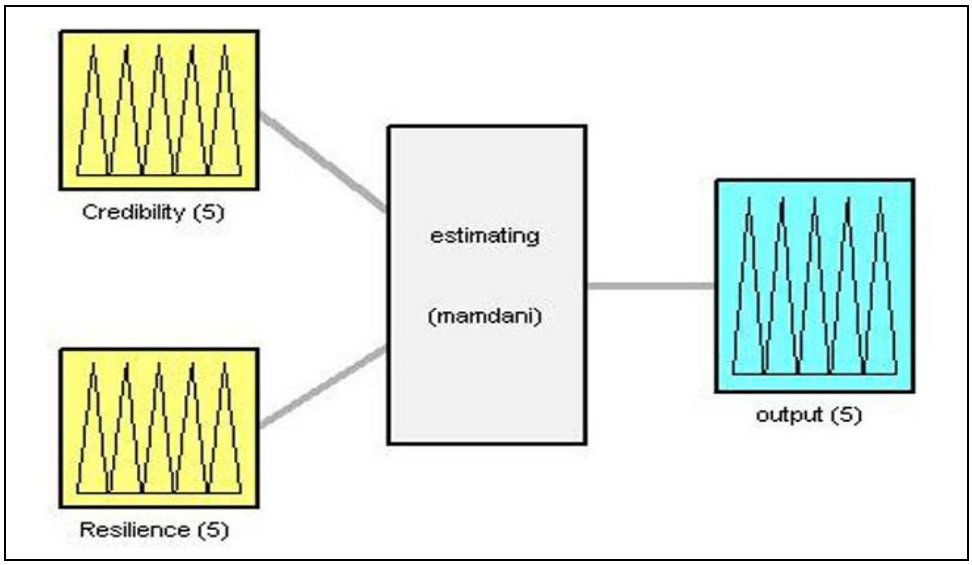

Figure 3.4. Proposed fuzzy inference system

3.6.2 Fuzzy Inference System Editor

FIS editor is used to display information about fuzzy inference system. FIS editor can simply be displayed by writing “fuzzy” in command windows in Matlab. Figure 3.5 show the graphical representation of FIS editor for our proposed method of estimation.

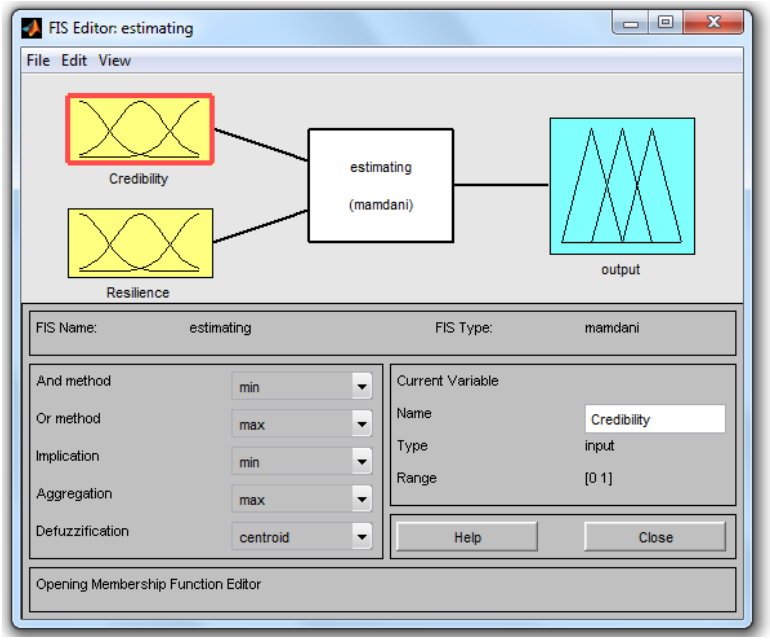

Figure 3.5. Graphical representation of FIS editor

3.6.3 Fuzzy Inference System model description

FIS is a system design, process the rules, and give results accordingly to the model developed for specific purpose (estimation) based on the rules in database. The designed FIS for estimation of software birthmark is in the form as;

fismat = readfis:('estimating')

The information about FIS will appear as

name: 'estimating'

type: 'mamdani'

andMethod: 'min'

orMethod: 'max'

defuzzMethod: 'centroid'

impMethod: 'min'

aggMethod: 'max'

input: [1x2 struct]

output: [1x1 struct]

3.6.4 Membership Function

MF is a curve, on which every input is mapped. It is the degree of extension of valuation. The values of MF is between {0, 1} interval. It is in the form $A=\{x, \mu_A(x) | x X\}$, where $\mu_A(x)$ is called MF of x in A. MF mapping each elements of x in the range of 0 and 1. The simplest MF is "trimf" function which is gathering of three points forming triangle and “trapmf” which has flat top and is condensed triangle curve. Figure 3.6 show the representation of how a membership function can be plotted.

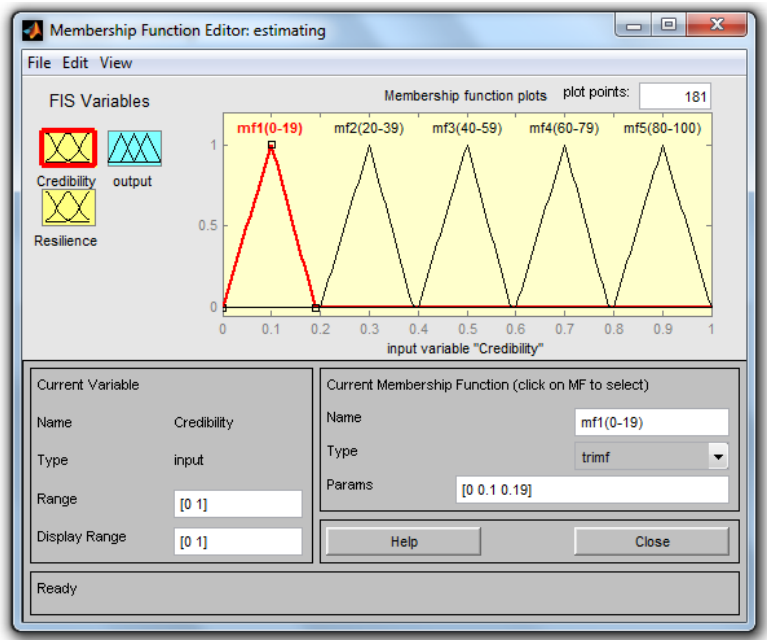

Figure 3.6. Membership function (input and output)

3.6.5 Mamdani Type Inference

Mamdani inference system is that type of FIS in which the fuzzy sets from the resulting of each rule are joined through the aggregation operator and the consequential fuzzy set is "de-fuzzified" the output of the system. In the proposed method Mamdani type inference system has been used, because it is mainly based on human input and also has extensive acceptance. Figure 3.7 shows the graphical representation of Mamdani and Sugeno type inference system.

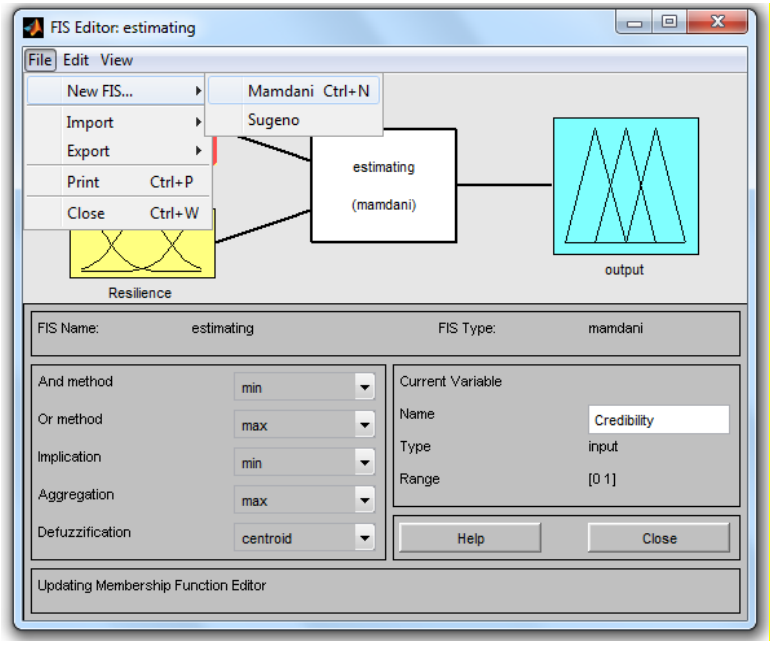

Figure 3.7. Mamdani type inference system

3.6.6 Sugeno Type Inference

Sugeno type inference is similar to that of Mamdani in many ways; in the first two part of Sugeno fuzzify the inputs by applying fuzzy operator. Sugeno type inference system is suitable for mathematical analysis. The difference is the Sugeno type output "mf" is either linear or constant.

Sugeno system lends itself to the use of adaptive techniques for constructing fuzzy models. These adaptive techniques can be used to customize the membership functions so that the fuzzy system best models the data.

3.6.7 Rules editor

The rules editor is used for the designing different rules based on the description inputs and outputs variables defined in FIS editor. Figure 3.8 show the rule editor for estimation of birthmark.

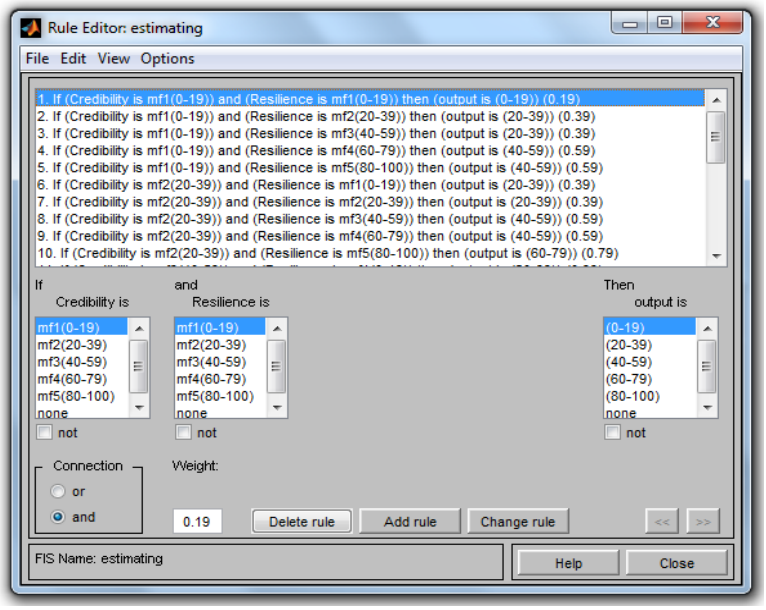

Figure 3.8. Rules editor for estimation of software birthmark

3.6.8 Rules viewer

When designed fuzzy rules can be graphically viewed through rules viewer. Figure 3.9 show the rules viewer for estimation of software birthmark.

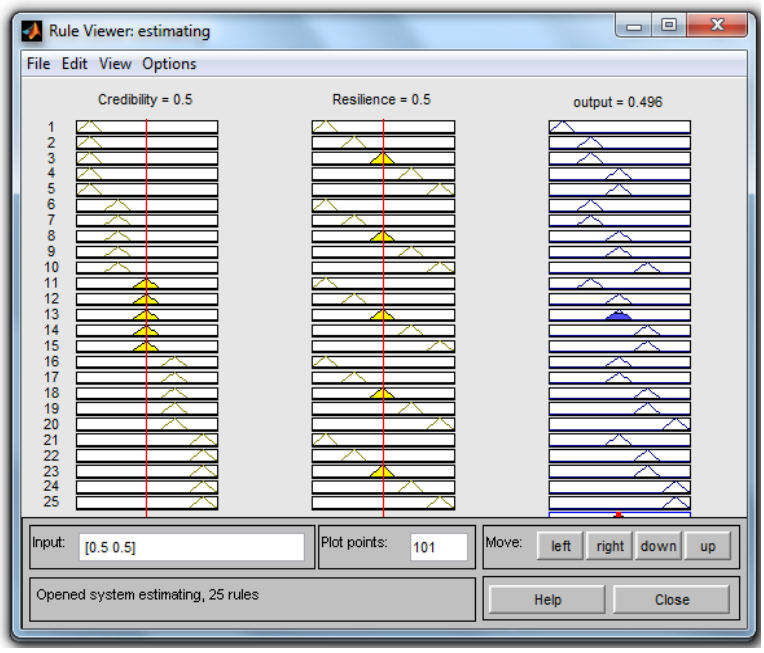

Figure 3.9. Rules viewer for estimation of software birthmark

3.6.9 Logical operators

Fuzzy logical reasoning is the superset of Boolean Logic. Commonly used logical operators are AND, OR and NOT.

3.6.10 IF THEN rules

The “IF THEN” rules used to devise uncertain description that comprises fuzzy logic. The “IF THEN” are formed as;

[IF "x" is A THEN "y" is B]

Here "A" and "B" are linguistics values in range "x" and "y" defined by fuzzy set. The antecedent is "x is A" of the IF part while the consequent "y is B" is the THEN part.

The rules of the proposed methodology for estimation of software birthmark are as follows;

If (Credibility is mf1(0-19)) and (Resilience is mf5(80-100)) then (output is (0-19)) (0)

If (Credibility is mf1(0-19)) and (Resilience is mf4(60-79)) then (output is (20-39)) (0.2)

If (Credibility is mf1(0-19)) and (Resilience is mf3(40-59)) then (output is (40-59)) (0.4)

If (Credibility is mf1(0-19)) and (Resilience is mf2(20-39)) then (output is (60-79)) (0.6)

If (Credibility is mf1(0-19)) and (Resilience is mf1(0-19)) then (output is (80-100)) (0.8)

If (Credibility is mf5(80-100)) and (Resilience is mf1(0-19)) then (output is (80-100)) (0.8)

If (Credibility is mf4(60-79)) and (Resilience is mf1(0-19)) then (output is (60-79)) (0.6)

If (Credibility is mf3(40-59)) and (Resilience is mf1(0-19)) then (output is (40-59)) (0.4)

If (Credibility is mf2(20-39)) and (Resilience is mf1(0-19)) then (output is (20-39)) (0.2)

If (Credibility is mf2(20-39)) and (Resilience is mf2(20-39)) then (output is (80-100)) (0.8)

If (Credibility is mf3(40-59)) and (Resilience is mf3(40-59)) then (output is (80-100)) (0.8)

If (Credibility is mf4(60-79)) and (Resilience is mf4(60-79)) then (output is (80-100)) (0.8)

If (Credibility is mf5(80-100)) and (Resilience is mf5(80-100)) then (output is (80-100)) (0.8)

If (Credibility is mf2(20-39)) and (Resilience is mf5(80-100)) then (output is (20-39)) (0.2)

If (Credibility is mf3(40-59)) and (Resilience is mf5(80-100)) then (output is (40-59)) (0.4)

If (Credibility is mf4(60-79)) and (Resilience is mf5(80-100)) then (output is (60-79)) (0.6)

If (Credibility is mf3(40-59)) and (Resilience is mf4(60-79)) then (output is (60-79)) (0.6)

If (Credibility is mf2(20-39)) and (Resilience is mf4(60-79)) then (output is (40-59)) (0.4)

If (Credibility is mf2(20-39)) and (Resilience is mf3(40-59)) then (output is (40-59)) (0.4)

If (Credibility is mf4(60-79)) and (Resilience is mf3(40-59)) then (output is (60-79)) (0.6)

If (Credibility is mf5(80-100)) and (Resilience is mf3(40-59)) then (output is (80-100)) (0.8)

If (Credibility is mf4(60-79)) and (Resilience is mf2(20-39)) then (output is (60-79)) (0.6)

If (Credibility is mf3(40-59)) and (Resilience is mf2(20-39)) then (output is (40-59)) (0.4)

3.6.11 Fuzzification of inputs

It is the procedure for generating membership values using MFs. The inputs in fuzzy logic are always crisp numerical values within the interval of 0 and 1, and the output is fuzzy degree of MF in choice 0 and 1.

3.6.12 De-fuzzification

In the de-fuzzification process the input is the aggregate output of fuzzy set. The cumulative of a fuzzy set encompasses a range of output values and has to be de-fuzzified to determine a single output value from the set. Five method used which are bisector, centroid, smallest of maximum, middle of maximum and largest of maximum [79]. The proposed model uses centroid calculation for de-fuzzification.

3.6.13 Customization

The fuzzy logic tool box is designed in such a way that gives freedom with the necessary limitation of the process illustrate and to modify the implication process of designed fuzzy inference. This provides open and effortlessly customized FIS structure.

3.7 Rules based approach to estimate software birthmark

Estimation of software birthmark is an essential part of software system development and maintenance to get rid of entire theft of the software system. Most of the software theft threats are faced during the implementation of the software. Developers are still in confusion how to handle such situations. If birthmarks of the system are estimated then one can easily make decision about the alternate design. The proposed methodology, based on fuzzy concept, provides an estimation model to software birthmark. Initially

inputs (properties of birthmark) are selected on the basis of which the birthmark(s) is to be estimated. On the basis of inputs the membership functions are plotted. The membership function identifies the degree of relationship of the concept (data) to a particular area (data range). Five membership functions were plotted that are mf1, mf2, mf3, mf4 and mf5. The inputs and membership functions are combined in rule editor which forms fuzzy rules. A fuzzy inference system model is then obtained based on membership functions and rules.

The idea of rule based estimation has been used by K. Tyagi and A. Sharma [83]. They measured the reliability of component based system. Fuzzy rules were designed to measure the reliability based on the four factors that are application complexity, reusability, component dependency, and operational profile.

3.8 Algorithm for designing a rule based model

The following are the steps to design the proposed model;

1. Perform domain analysis on software birthmark
2. Identify properties of software birthmark on which birthmark is to be estimated
3. Establish an input data base for these properties
4. Design the fuzzy inference system based on these properties (inputs)
5. Define the membership functions for these properties (both for inputs and output)
6. Design the fuzzy rules based on membership functions
7. Obtain a fuzzy inference system (model to estimate birthmark)
8. Estimate the inputs accordingly.

The graphical representation of the algorithm is given in figure 3.10.

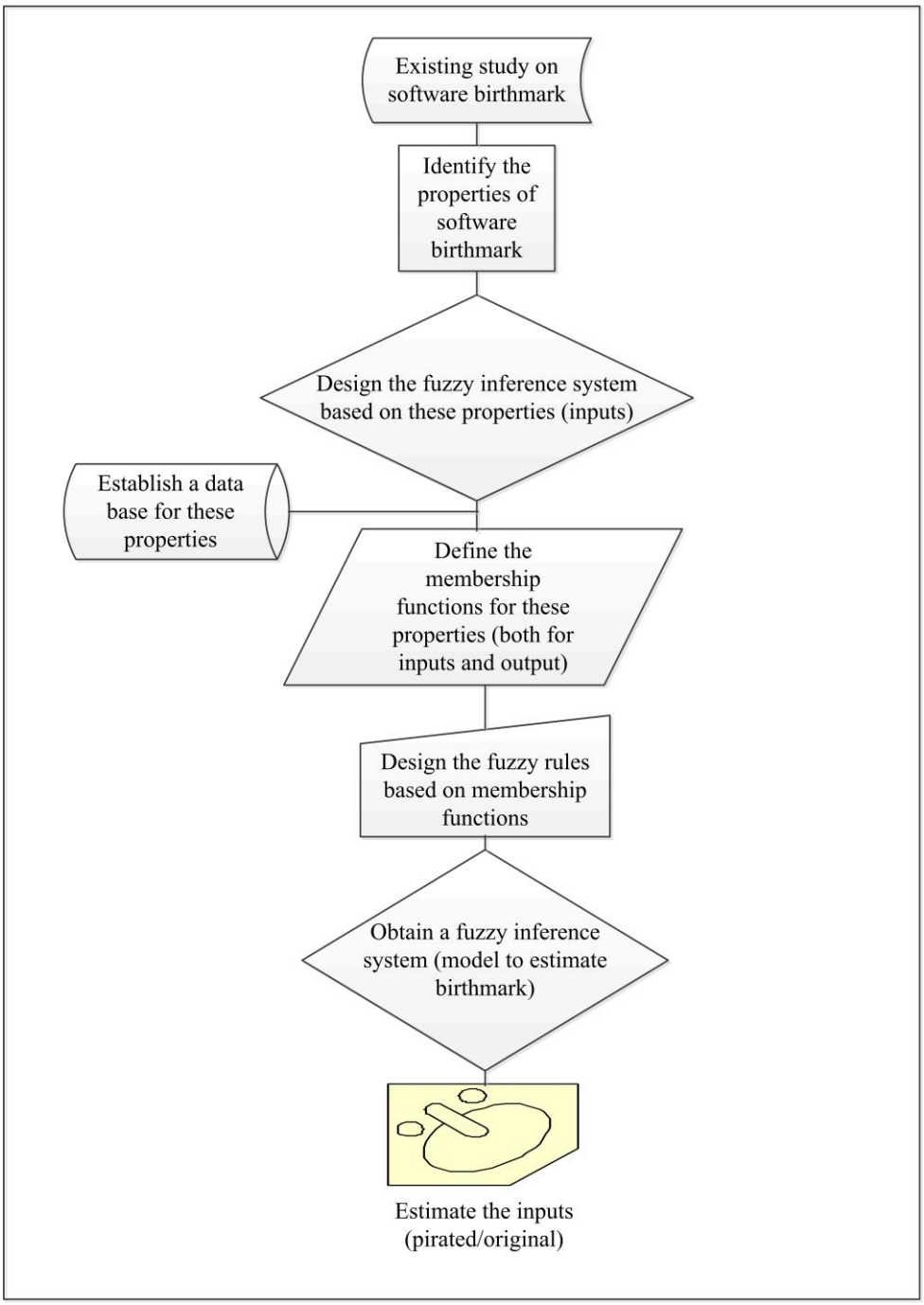

Figure 3.10. Proposed algorithm for rules based model

The proposed work for estimation of software birthmark has been carried out by using Matlab fuzzy tool box [84].

The different membership combinations are given below in table 3.1.

Table 3.1. Membership function pairs

mf 1, mf 1	mf 1, mf 2	mf 1, mf 3	mf1, mf 4	mf 1, mf 5
mf 2, mf 1	mf 2, mf 2	mf 2, mf 3	mf 2, mf 4	mf 2, mf 5
mf 3, mf 1	mf 3, mf 2	mf 3, mf 3	mf3, mf 4	mf 3, mf 5
Mf 4, mf 1	mf 4, mf 2	mf 4, mf 3	mf4, mf 4	mf 4, mf 5
mf 5, mf 1	mf 5, mf 2	mf 5, mf 3	mf5, mf 4	mf 5, mf 5

Linguistic variables used as a fuzzy set {very low, low, medium, high and very high} = {VL, L, M, H and VH} are plotted in the area under the range of 0 and 1. The estimation of input was based on the concern skill by means of expert opinions. There are "5" MF and "2" inputs, so a total of 32 rules was designed. After plotting membership functions and designing the rules, a model of fuzzy inference system is obtained. Inputs are given to the designed model and evaluated for the purpose of estimation of software birthmark in term of credibility and resilience. Based on the results obtained from the fuzzy inference system, the decision regarding the software birthmark can be made that either the birthmarks of the software are same or not.

The fuzzy rules and model in the proposed methodology are given below in figure 3.11.

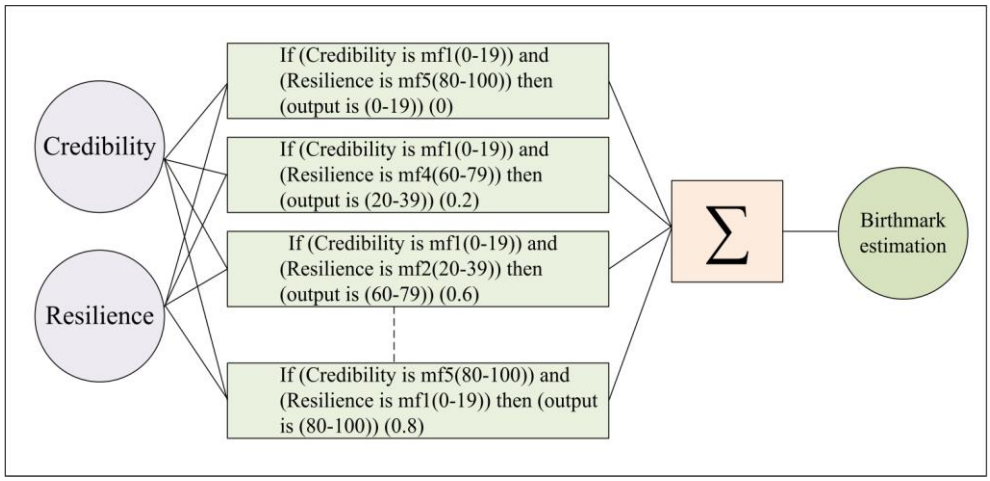

Figure 3.11. Proposed fuzzy rules model

The proposed model can further be explicitly explained below in figure 3.12. In this figure the dark large circle shows inputs and outputs. The second large white circles show membership functions and the middle dark circles shows the rules.

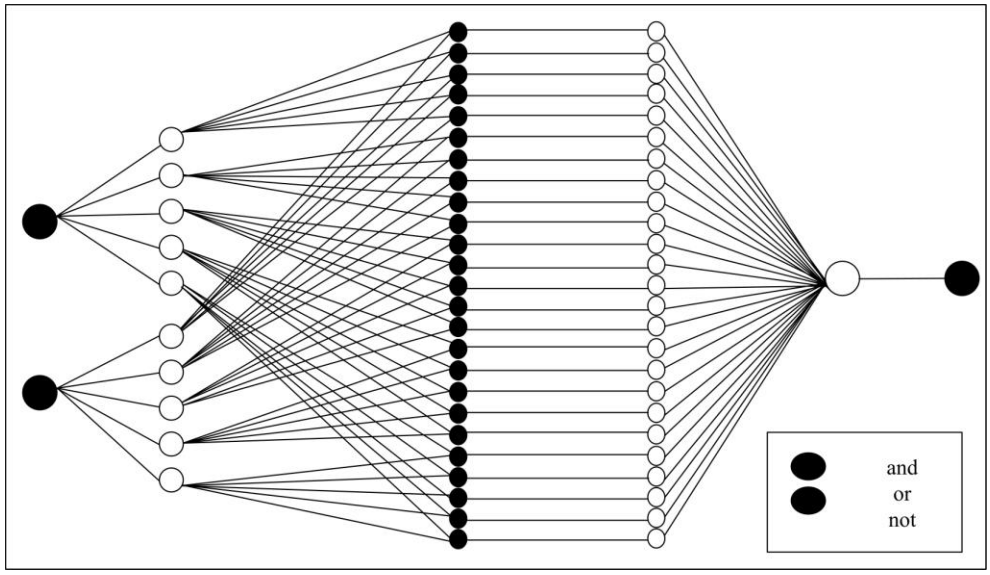

Figure 3.12. Graphical representation of rules model (inputs, membership functions, rules and output)

Based upon the above rules a fuzzy inference system is obtained for estimation of software birthmark. Figure 3.13 visually shows the surface view of inputs and output.

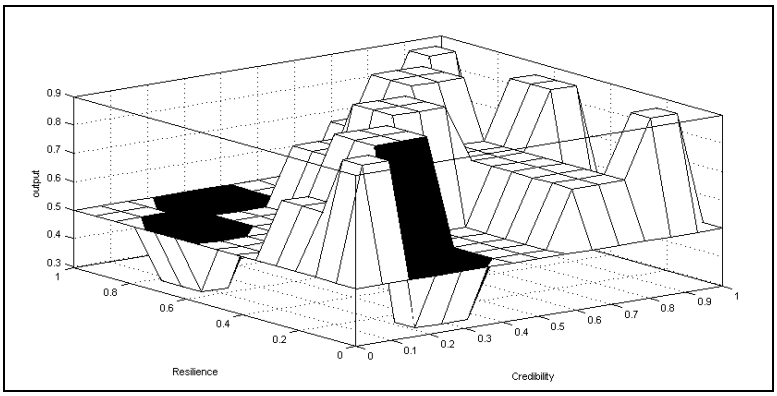

Figure 3.13. Surface view of inputs and outputs (generated in Matlab)

3.9 Input estimation

Once the fuzzy rules model is designed, inputs will be given according to the user requirements to the model. The model will generate the output based on the fuzzy rules. Details of the proposed system, inputs, and output are given in table 3.2.

Table 3.2. Proposed model (inputs and output)

Model	[System] Name='estimating', Type='mamdani', Version= 2.0 NumInputs= 2, Num Outputs= 1, And Method= min Or Method= max, Imp Method= min, Agg Method= max Defuzz Method= centroid
[Input1]	Name= ‘Credibility' Range= [0 1], Num MFs =5 MF1= mf1(0-19) trimf, [0 0.1 0.19] MF2= mf2(20-39) trimf, [0.2 0.3 0.39] MF3= mf3(40-59) 'trimf, [0.4 0.5 0.59] MF4= mf4(60-79) trimf, [0.6 0.7 0.79]

	MF5= mf5(80-100) trimf, [0.8 0.9 1]
[Input2]	Name='Resilience' Range= [0 1], Num MFs=5 MF1= mf1(0-19) trimf, [0 0.1 0.19] MF2= mf2(20-39) trimf, [0.2 0.3 0.39] MF3= mf3(40-59) trimf, [0.4 0.5 0.59] MF4= mf4(60-79) trimf, [0.6 0.7 0.79] MF5= mf5(80-100) trimf, [0.8 0.9 1]
[Output]	Name='output' Range= [0 1], Num MFs=5 MF1= (0-19) trimf, [0 0.1 0.19] MF2= (20-39) trimf, [0.2 0.3 0.39] MF3= (40-59) trimf, [0.4 0.5 0.59] MF4= (60-79) trimf, [0.6 0.7 0.79] MF5= (80-100) trimf, [0.8 0.9 1]

3.10 Evaluation of the model (Case study)

The proposed model is validated by a case study of small module for Android application. The Android "radiocalc" module consists of 109 lines of code. The methodology has been applied on the similar application for Android. The birthmark of the module has been estimated based on the properties of resilience and credibility.

K-gram based birthmark similarity technique [66] has been used. By performing various experiments we found out that as the K-value increases the birthmark similarity

decreases. For very small values of K- the birthmark similarity was not satisfactory. For k= 5, the experiment revealed good results in term of similarity and runtime overhead. The resulted similarity for the above mentioned application with k= 5 was 40 %.

We applied SandMark [72] and Codeshield [85] tools for the above application for code obfuscation. To find the value of resilience it gives a similarity of 80% for k=5. Codeshield provides the name obfuscation, the removal of debugging information, and some type of control flow, while the SandMark does not include an automatic obfuscation. The similarity was computed through K- grams. The similarity of Codeshield was found for K- gram, which shows that if K increases, there is a decrease in the similarity for numerous of the transformations. The following table 3.3 shows the inputs and values for the proposed model.

Table 3.3.Inputs and value for the proposed model

Inputs	For k= 5	
	Value in %	Value for proposed model
Credibility	40%	0.4
Resilience	80%	0.8

The defined inputs to the fuzzy model are described as; If credibility = 0.4 (40%) and resilience is 0.8 (80%). These inputs are given to the fuzzy inference system. Credibility 0.4 is the degree of membership function mf1 (40-59) and resilience 0.8 is the degree of membership function mf2 (20-39). It will give the output 0.50 from the degree of membership function based on the designed model. The output below 0.5 show that software has low level of piracy, while the output above 0.5 show that software is highly

or completely pirated. So from the results one can make a decision about the birthmark of the software.

3.11 Results and discussion

A fuzzy inference system is designed which models the system which in turn estimates the birthmark of the software. Inputs assign to the model to check and estimate the software birthmark in term of credibility and resilience. The designed model evaluates the inputs (which are given to the model) and give results. On the basis of the given results one can check the estimation of software birthmark for the properties of credibility and resilience. To check the validity of the proposed model inputs were given as; out=evalfis ([0.4 0.8], fismat), the output = 0.500, which show the estimation of the software birthmark. The output near to "0" show that software has low level of piracy, while the output near to "1" show that software is highly or completely pirated. Hence, this result clearly shows the software birthmark for their desired properties.

Summary

This chapter includes the complete details of the proposed methodology for estimation of existing software birthmark based on two most important properties, which are credibility and resilience. The concept of fuzzy logic has been used as the main methodology for designing fuzzy rules for the estimation of software. The chapter explains at length the concepts related with the proposed methodology including explanation of rule based approach, estimation, use of estimation of birthmark, algorithm for designing a rule based model, and also evaluation of the model.

Chapter 4

Identification of features as softawre birthmark

Software can be dissected into features under various categories, such as syntactic features and semantic features that contain all the information related to the construction and functionality of the software. These features have intrinsic connections in-between which uniquely identify their working in a particular piece of software. These characteristics of software are known as a birthmark. A lot of research has been conducted to identify different techniques to define software birthmarks. A software birthmark uniquely identifies software and hence used to detect software theft and piracy. This research aims at identifying as much software features as possible and proposes a software feature model based on features lying under different categories. Each of the software will possess a unique value set for features identified according to that feature model. These value sets can then be used to detect similarity among software programs.

4.1 Software features and theft detection

A software program is a collection of different software features of certain types. A clear understanding of these features and their organization into logical categories is another step further in understanding the code. This understanding of a specific program code eventually helps in identifying similarities among more than one instance or copies of presumably same software application (that is, the program). The identification of similarities hence facilitates piracy and software theft detection. Some frequently used techniques for defining software birthmark based on one, two or a small set of features

are already identified. Some of the techniques are applicable on program source code while others are meant to be used only with byte-code.

There are different categories under which software features can be placed. For example, functional features, which relate to the functional requirements of the user, for example, calculating profit on sales; structural features, which relate to the inner structure of the software, for example, number of functions in the software program; quality features, which are associated with the quality requirements of the software, for example, ease of use and reliability, etc. Y. Guo et. al [17] provide a categorization as input software features, self-software features and output software features. Silvio and Yang [18] categorized the features into syntactic and semantic features. Syntactic features deal with the structure of the program, while semantic features deal with the meaning of the program. Figure 4.1 shows the different features of a program.

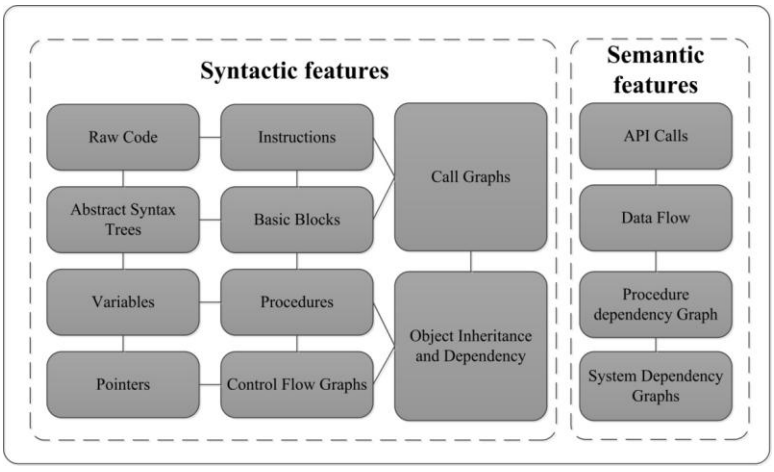

Figure 4.1. A taxonomy of software program features [18]

According to K. C. Kang et.al [58] different program features are processed at different phases of a software program. In this sense features can also be categorized and identified as compile time features, load time features and runtime features.

This fine grained categorization of software program features provides a metaphor to identify an exhaustive set feature for any software system. These software features provide a lot of important information about the software system that they present. At the same time the software features that uniquely identify a software system are used in critical operations, for example in software theft detection. Several techniques related to software theft are used by researchers and industry, for example, program identification for software theft [68], static API trace based detection [5], behavior based software theft detection [86], static instruction trace based theft detection [4], heap graph for software theft detection [87], and customizing k-gram based birthmark for software theft detection [26], etc. All these techniques are based on some unique program features termed as software birthmark.

4.2 Similarity measurement of software birthmark

S. Cesare and Y. Xiang [18] worked on software birthmark similarity measure for different classes of birthmark. This similarity measures include distance metrics, string similarity, vector similarity, set similarity, set of vectors similarity, tree similarity and graph similarity. The distance metrics specifies that searching and indexing in a database become easy, if there is a distance metric in a distance function. String metric can be used for comparing string metrics to show the similarity. The vector distance can be measured by using different metrics such as Euclidean distance or Manhattan distance. Set similarity is another types of similarity checking of software. Two sets can be compared by using set similarity. The set of vectors similarity can be compared using the minimum matching distance. Trees similarity can be used for comparing equality by using tree

isomorphism. Graphs similarity can be used for structure equality by using graph isomorphism.

The similarity of birthmark can be measured by finding the resemblance among them. Suppose f(p) = {p_1, p_2, . . . p_n} and f(q) = {q_1, q_2, . . . q_m} be the birthmark of modules p and q. In this situation both of the sets are same if f(p)= f(q). Broder [88] presents a similar idea for comparing the files. Two mathematical notations that are resemblance and containments were defined to measure the similarity of documents.

The resemblance of file *p* and *q* is defined by the formula;

$$r(p,q) = \frac{\left| f(p) \cap f(q) \right|}{\left| f(p) \cup f(q) \right|} \qquad (4.1)$$

And the containment of file *p* and *q* is given by the formula;

$$c(p,q) = \frac{\left| f(p) \cap f(q) \right|}{\left| f(p) \right|} \qquad (4.2)$$

Here $\cap$ and $\cup$ operations are set union and intersection operations, and || denotes set cardinality. One of the following scenarios can be considered while measuring the similarity of two programs *p* and *q*. The concept is presented in figure 4.2 [89].

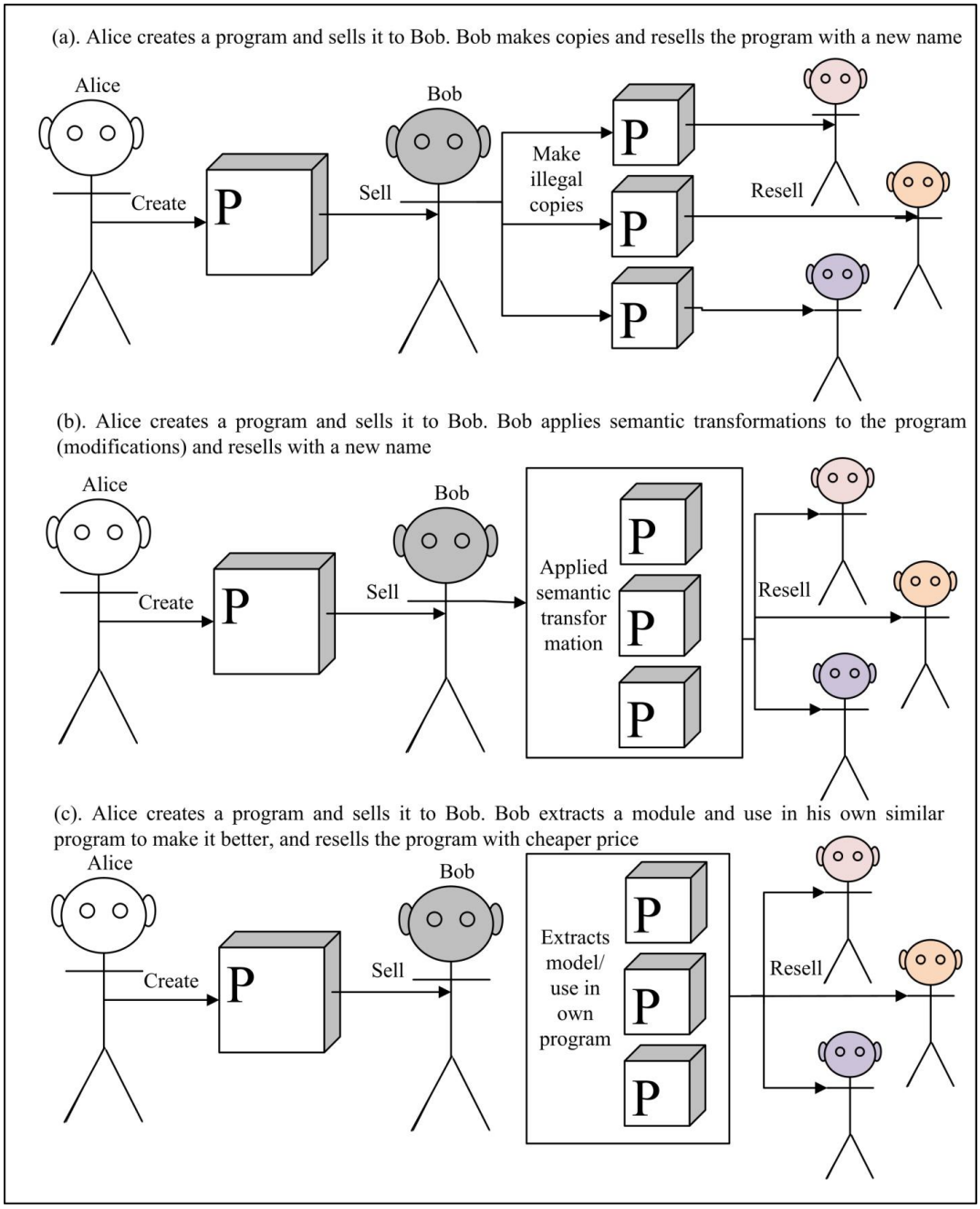

Figure 4.2. Representation of different types of piracy

4.3 Dissection and analysis of software features as a birthmark

In this section a software feature model is presented which is based on an exhaustive list of features that can be identified in a software program. Taxonomy of the categories of software features is presented in figure 4.3. The taxonomy is designed keeping in view

the generic sequence which is normally followed to analyze a software program. This sequence also presents the inherent relationships among different feature categories.

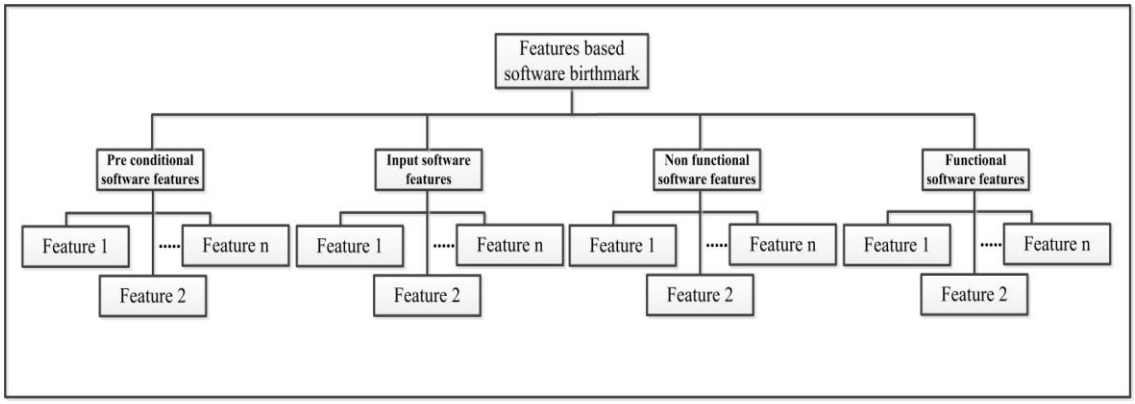

Figure 4.3. Representation of software features

Figure 4.3 shows the software feature model which is based on the taxonomy presented in figure 4.4. The model groups together related software program features under four broad categories, named as pre-conditional software features, input software features, non-functional software features and functional software features. A software program can be analyzed to identify (or to calculate) specific values for these features. Each of these features may be assigned a quality value (for example, good, high, etc.) or a constant value (for example 10, 25, etc.) depending upon the type of feature (either qualitative or quantitative in nature). The resulting value set of a software program will be the birthmark of the software. The value sets of candidate software programs can then be compared to find out similarities and to detect software theft. Figure 4.4 shows the details of identified features as birthmark.

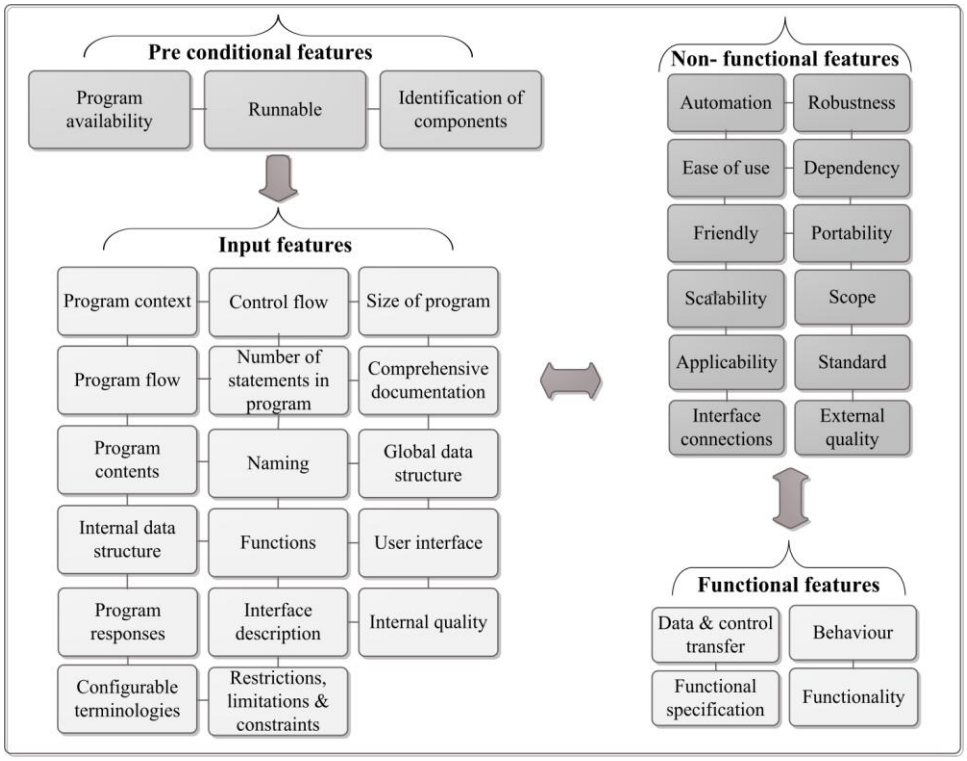

Figure 4.4. Features as a birthmark

Figure 4.5 shows the proposed process of comparing similarity of two programs. The process starts by analyzing the software for feature extraction. Different techniques may be applied for feature extraction depending upon the type of features. After the extraction of features and their values the candidate programs will be compared. If the similarity among features is found above a predefined threshold value then the software programs will be termed as similar otherwise dissimilar. In this way value sets (based on software feature model) of different software programs may be compared with the value set (based on software feature model) of the original copy of software program to detect piracy and theft.

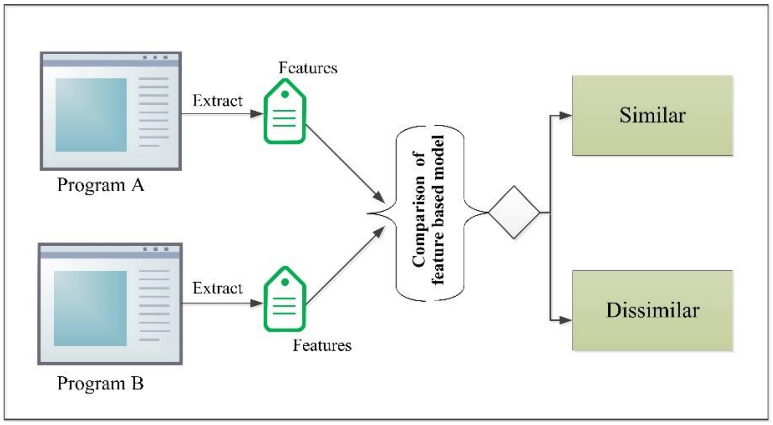

Figure 4.5. Program similarity checking

The success of the proposed process depends upon the identification of a considerable set of features which may be identified in an organized manner for a software program. The model is based on the collectiveness of features means the feature(s) independently cannot act as a birthmark. The following sections briefly define the identified software features under their specific category.

4.4 Pre conditional features

Details of the pre conditional features of a software program are given below.

4.4.1 Program availability

The first and the most important feature in detecting the similarity of software or programs is the availability of the original software program (and the candidate software which is to be analyzed for theft detection). The availability of software is that stage at which all the essential actions are carried out and the software become accessible. The program availability should be logically complete so as to be able to analyze it properly

for the purpose for which the analysis takes place. Mathematically software availability (software is working) A(t) at the time "t" is shown as in [90].

$$A(t) \triangleq P\{I(t) = 1\} = \sum_{k=0}^{n} P(k, t; n), \quad t \geq 0 \qquad (4.3)$$

And the software is not working (software unavailability U(t)) at time "t" is mathematically shown as;

$$U(t) = P\{I(t) = 0\} = \sum_{k=1}^{n} Q(k, t; n), \quad t \geq 0 \qquad (4.3)$$

4.4.2 **Runnable**

When a program is to be checked for the extent of piracy, it should be runnable. An executable program is needed to properly analyse its relevant features for the purpose of checking piracy.

4.4.3 **Identification of components**

According to G. Caldiera and V. R. Basili [91] a software component "is simply a container for expressing abstractions of data structure and algorithms". The components of a system are the building block and may be different. Several diverse approaches to component identification are already been published in literature. These approaches make use of different component definitions and identification strategies [92]. The software component can be divided into "Business", "Logical" and "Technical" component categories. Different components inside the program are identified for feature extraction. A detailed description of each component should be contained within the structure of the program. Whole set of components and their interface should be clearly identified so that

features can be traced in the program. Reusability and maintainability are the two significant factors in the identification of component [93].

4.5 **Input features**

Input features category contains the following sub features.

4.5.1 **Program context**

The context information and details of the program can be defined. It requires information about primary inputs and outputs. The “big picture” of the program should be defined in order to clearly understand the program. The program is placed in the context of product and business (domain). Identification of the context also adds another attribute to detect similarity in software programs.

4.5.2 **Program flow**

Program flow refers to the order in which the program components execute. Flow of a software program can be identified with the help of data flow diagram, state transition diagram, and control flow diagram to check the similarity among software programs. A program’s sequence of flow can be compared with the other program sequence of flow that can present level of piracy.

4.5.3 **Program contents**

The contents of a program are set of statements and functions. It includes all program code organized in a defined structure to create functional modules (method, classes etc.) performing certain required operations. Comparison of programs on the basis of their contents also shows similarity among programs.

4.5.4 **Internal data structure**

Internal data structure of a program is passed among different components or modules of the program. The internal data structure also refers to the flow of data in a proper sequence. Two programs having same internal data structure also shows the similarity among programs.

4.5.5 **Program responses**

The responses are outcome of a program. Different modules or sub programs interact with each other. For a functioning program it requires proper implementation along with an interaction strategy through which different units of a program can interact to ensure that features are working in the way it is required [94, 95]. Similarity can also be detected on the basis of the output being generated by programs. Exactly same output also contributes to a degree of similarity among software programs.

4.5.6 **Configurable terminologies**

Configurable terminologies are the relevant terms associated with a program. It includes what types of inputs will be passed through a program? What operations will be performed by the program? And what results will be displayed by the program? The terminologies of a program can be compared with the terminologies of another program to check for piracy.

4.5.7 **Control flow**

The control flow of a program refers to the control order in which the program executes. Control flow breaks up the flow of execution of a program by employing decision

making regarding the aspect for which the control is imposed. Within a program different types of control flow are executed for different functionalities. These control flow can be executed in individual statement, instruction or function call. The control flow of the programs can be checked for finding similarity in the programs.

4.5.8 **Number of statements in program**

A statement is the smallest part of a program which expresses some action. It is the instruction, input to the system for performing some action. These are static characteristics which can be identified using static analysis. Lines-of-code is a most commonly used static metric which will also help in identifying similarity.

4.5.9 **Naming**

The naming (variable name, functions names, etc.) used in one program can be compared with the naming of another program to check the extent of piracy. This naming differs from the configurable terminologies where configuration of terminologies is involved.

4.5.10 **Functions**

A function is module of code which performs a specific well defined task. Functions usually take data as input, process it and give result(s). A function performs some task which is useful for other parts of the program. While other parts of the program does not need the detail of how the function is implemented. A function of one program can be compared with another program to check its behaviour against piracy.

4.5.11 **Interface description**

The interface is the way of communication between components in a program. Each component has two interfaces; provide interface and required interface. Provide interface

defines the services that are provided by a component for the other components, while required interface define the services that specify what services must be made available for this component for its proper working. These interfaces of a program can be checked and compared with that of another program to show the level of piracy between the programs.

4.5.12 **Restrictions, limitations and constraints**

Restrictions, limitations and constraints are the principles which limit the accessibility of a program. Software restriction policies can be applied in the form of an allow list or deny list. The allow policy of the system is restricted by default and blocks all the applications that are explicitly listed as a restricted. While in the deny policy the default rule is unrestricted and restricts those applications which we explicitly mention to be restricted. Limitation is a condition of bounding a program. It is also a principle of limiting the scope of program. Constraints are restriction on program. Constraints are effectively global requirements, such as limited development resources, organizational policies etc. These principles applied on one program can be compared with the principles applied to other program to check the similarity among them.

4.5.13 **Size of program**

The size of a program on disk can also be calculated and provides useful information. A program with more code will ultimately results in large size of the program. Under execution, it tells us how much is the size which usually it takes in memory during its operation. The size of a program can also be measured as physical measurement that include lines of code (LOC), kilo lines of code (KLOC) non commented line thousand LOC (NKLOC) and logical lines of code (LLOC). Several other measurements related to

software can also be performed. Such measurements include language productivity factor, counting reused and refactored code, counting nonprocedural code length, measuring the length of specialization and design etc. [96].

4.5.14 **Comprehensive documentation**

Comprehensive documentation provides a detailed description about the program in which all the relevant information is provided. It presents precise and usable documents which helps one to identifying the similarity between the programs.

4.5.15 **Global data structure**

Global data structure is the structure of data which is available in the major parts of the program. It can be compared with another copy or with the same program for checking the similarity.

4.5.16 **User Interface**

UI is the way of communication of program with the user. It is the most important way and part of the program through which the program can easily be accessed. The goal of user interface is to allow effective operation and control the program from human perspective and make the interaction easy, efficient and user friendly. Similarity among programs can also be detected by analysing the similarity in UI components of two software programs being checked for similarity.

4.5.17 **Internal quality**

Internal quality deals with the internal structure of the program and is about the design of the software. Internal quality is used to facilitate the process of a good and reliable product. It includes cohesion, low coupling, simplicity, generality and clarity. Internal

quality characteristics are maintainability, flexibility, portability, re-usability, readability, testability, and understand ability. These features of a program can be compared with the same features of another program to show the level of similarity.

4.6 **Non-functional software features**

The software requirements are specified in the field of domain engineering before the software is developed. The software functional requirements are easy to describe and implement. While the nonfunctional requirements are frequently not clear that how to implement when there are multiple components in the program. The non-functional software or program features are those features which are indirectly provided by a software program. These features sometime depend upon the input features of a software program. Non-functional features are often called qualities of a system. Non-functional features include availability, efficiency, flexibility, portability, integrity, performance, reliability, reusability, robustness, scalability and usability. For a software system the nonfunctional properties can also be measured and it is needed how to measure the individual property [95]. Further details of the non-functional software or program features are given below.

4.6.1 **Automation**

Automation software provides effective automated data acquisition and control systems. It is the use of control for operating the system. The control of original and pirated (copy) program can be checked for the purpose of checking duplication in the program.

4.6.2 **Ease of use**

The ease of use of a program shows how easily a program can be used. Ease of use also refers to the term usability. In ease of use, the user requirements are understood, for formulating the usability goal and to evaluate the usability of the system. The usability goals of programs can also be compared to check the similarity in programs.

4.6.3 **User friendly**

A good software or program is the one which is user friendly. User friendliness depends upon different characteristics. The evaluation of these characteristics demonstrates the user friendliness of a program. User friendly software(s) can provide a good user experience. A user friendliness software program has the feature of simplicity, clear and intuitive interface, even if it is complex software. User friendly software is more successful than those of complex software which is having complicated interface. The software industry performs user testing of software before releasing in the market. Two programs can also be compared on the basis of their defined user-friendliness.

4.6.4 **Scalability**

The scalability is the ability of a program to evolve in order to meet customer requirements. Scalability of software plays an important role in the software business. As the software is growing due to its large scale usage, the scalability of software makes it easy to upgrade new user requirements. Scalable software should grow more efficiently as more demands are place on the software. The scalability of a program can be compared with the scalability of another program to check the extent of piracy.

4.6.5 **Applicability**

The programs should be checked in the context of application domain for which they are developed. The applications of the program can be checked for which it is designed. If one of them is different from another, then both the programs are different.

4.6.6 **Interface connection**

The interface connection is a specific connection in which the interfaces of the program are connected with each other forming a complete software or program. A connection represents a specific session with interfaces. An interface connection is able to provide information describing the connection. Interface connection of both the original and copy (same) programs should be checked to show its piracy

4.6.7 **Robustness**

The robustness of a program is the ability to handle errors during execution. Robustness is a nonfunctional feature of software which should be designed into the software from the start. Robustness can also be measured empirically.

4.6.8 **Dependency**

In software dependency the different features of software and their interaction are dependent upon on each other. This dependency can be affected by human and organizational factors to how it can be handled in term of software quality. The dependencies can be formed through the analysis of source code or byte code. The relationship of a software dependency can be represented either by data related dependency or by functional dependency. The present research has focused on a single dependency type that can be syntactic or logical for the relationship of failure proneness

to the dependencies of software. Further the research has also focused on the human and organizational factors that are based on quality for the failure proneness of the software.

4.6.9 **Portability**

Portability is the program or software of the same functionality that is adopted or produced in more than one place. The portability of a program is the usability of that program in different surroundings. The portability of a program can be compared with the portability of another program to check whether the program is pirated or not.

4.6.10 **Scope**

The scope of the software or program is the activity which limits the program by defining behaviour. The scope documentation lists explicit program goals, deliverables, tasks and deadlines. Inclusion and exclusion criteria for functional requirements are the main part of scope that what to include in the software and what to exclude from the software. Defining the scope of a program is important because later on it cannot be changed once if it is already defined, although it can be upgraded. The changes of scope typically create a lot of problem for customer as well as for developer. The scope of both the original and copy or same program can be compared to checking the piracy in the programs.

4.6.11 **Standard**

The software or program standard enables interoperability among different programs developed by different developers. Software or program standards having definite terms, concepts, data, formats, styles of documents and different techniques. Some of the standards are controlled by an authoritative body such as IEEE, ISO, and ACM etc. Both

the original and pirated programs can be checked for standards we detecting the level of piracy.

4.6.12 **External quality**

In order to measure the success of software program, it is worth that software or program must have the internal and external quality. The external quality is the property of a software or program that a user faces and experiences it. It means the system is providing the required functionality or not. The system has a clear and user friendly interface or not? External quality of a system is also based on internal quality of the system. It includes conformity, reliability, accuracy, correctness, easy to use, adoptability and robustness, etc.

4.7 **Functional software features**

Feature interactions play a key role in the functionality of a software system. To work correctly, a program not only requires the implementation units that communicate to the selected features, but also an interaction unit that ensure that features operate together in a specified way. The software system functionality is divided into inputs, output, internal data files, external interface file and the related processes. A system is said to be feature-rich when it has adequate number of options and functional capabilities available to the user

4.7.1 **Data and control transfer**

The data inside the program can be transferred from one part of the program to another part by calling some set of statements. On the other hand, control transfer defines the flow of execution.

4.7.2 **Functional specification**

The functional specification is the formal and essential requirements (document) regarding program which clearly describe the important requirement and capabilities of a program. It is also the documentation which describes the behaviour of a program. The functional specification depicts what is needed to the program and what are the required properties of inputs and outputs. A functional specification is a detail technical response to the respective requirements documents. The specification helps a program in finding the relevant terms related to a particular program. For example to estimate functionality of a program, we have to define all the relevant terms related to a program in the specification. The functional specification of a program is a set of guidelines that provide an accurate and efficiently estimate the cost of design alternative. It also includes formal description of a user task(s), dependencies on the other products and the criteria of usability. The functionality of both the programs is compared to check the extent of piracy in programs.

4.7.3 **Behaviour**

The behaviour is the action of a software or program. To understand software behaviour can help user in various aspects and task of the software. The behaviour of programs can be compared to find the similarity in software.

4.7.4 **Functionality**

The functionality of a program is the aspect of what a program or software can do for a user. The software can not only be measured physically. It can be measured based on its functionality. Users and customers care about the functionality and not how many lines of code it is.

These overall features of software can be represented as;

Overall features= {PCF+IF+NFF+FF}. PCF is the pre conditional features, IF is input feature, NFF is non functional features, and FF functional features.

Summary

This chapter is based on the features of a software system that are pre conditional features, input features, non functional and functional features. These features are having further a total of 36 different features, which are identified for the purpose of checking similarities of software in term of birthmark of software. These 36 features (birthmark) of software can be compared with the duplicate copy or another software for checking the piracy in software.

Chapter 5

Estimation of software features based birthmark

Software birthmark is a property of software that can be successfully used to detect piracy and theft. A birthmark based on a number of software features can provide an even close estimation and detection of software piracy. The estimation of a birthmark can play a key role in proving that the birthmark is the true unique identifier of the software under study. In this chapter the concept of fuzzy logic has been used to estimate the credibility and resilience of different software features based birthmark. The following sections describe the methodology portion of the proposed research work carried out in this chapter.

5.1 Software features identification

Software feature contains all the essential information of a software system. Features are the static attributes and information about functional and nonfunctional qualities that are present in any software system. These software features are almost interlinked with each other, performing different operations and due to these operations the software or program is considered to be a functional software system. An obvious understanding of these software features and their association into logical categories is an additional step to further understanding the program code. This understanding of a specific program code can ultimately help in identifying the similarities among software application. Feature interactions play a key role in the functionality of a software system. A software or

program is said to be feature-rich if it has many functional capabilities available to the user.

The software requirements are specified in the field of domain engineering before the software is developed. The software functional requirements are easy to describe and implement. While the non functional requirements are frequently not clear that how to implement when there are multiple components in the program. The non-functional software or program features are those features which are indirectly provided by a software program. These features sometime depend upon the input features of a software program.

5.2 **Software birthmark estimation**

The estimation of birthmark can play a vital role in accepting the effectiveness of a birthmark. There is a need for a platform and design independent definition of software birthmark, along with a formal estimation model, to facilitate software industry in detecting software piracy and theft. Features estimation with best accuracy helps in detecting and identifying software theft or piracy. Before estimation we cannot define whether the software is original or pirated.

Comparison of birthmarks is essential for checking similarity of software programs. If the birthmarks of software are similar, ultimately the software programs are similar. Features of a software program are considered to be a birthmark and can be compared with the other birthmarks of the software program to show the extent of originality and similarity of the software program. Figure 5.1 visually shows the different features of a software program.

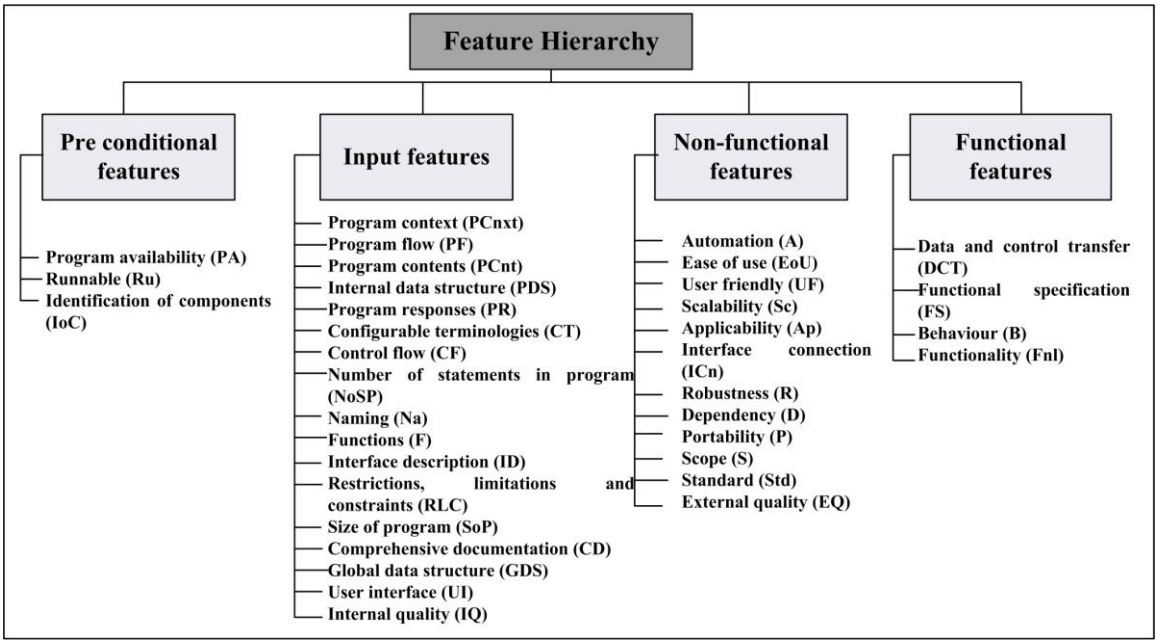

Figure 5.1. Software features

5.3 Fuzzy logic

Fuzzy logic is a mathematical tool used for solution of uncertain and vague data. Fuzzy logic was developed and used by A.Z. Lofti in 1965 [78]. It copies the human reasoning process, measures imprecise information and provides a best decision in the presence of the vague and incomplete data. Fuzzy logic has a wide range of applications in real life, such as control system, car transmission system, washing machines and vacuum cleaner etc [97]. The fuzzy logic concept is mostly used when only the subjective information is available. The fuzzy expressions are more natural. The proposed Fuzzy inference system makes it easy to build facts and provide solution for unknown information. Further details about fuzzy concept can be find in Zadeh [78].

In the proposed method, different categories of features have been used as input to the fuzzy inference system. These inputs involved preconditioned features having sub features of Program Availability (PA), Runnable (Ru) and Identification of Components

(IoC). The input features having sub features of Program Context (PCnxt), Program Flow (PF), Program Contents (PCnt), Internal Data Structure (PDS), Program Responses (PR), Configurable Terminologies (CT), Control Flow (CF), Number of Statements in Program (NoSP), Naming (Na), Functions (F), Interface description (ID), Restrictions, limitations and Constraints (RLC), Size of Program (SoP), Comprehensive Documentation (CD), Global Data Structure (GDS), User Interface (UI) and Internal Quality (IQ). The Non-functional software features having sub features of Automation (A), Ease of Use (EoU), User Friendly (UF), Scalability (Sc), Applicability (Ap), Interface Connection (ICn), Robustness (R), Dependency (D), Portability (P), Scope (S), Standard (Std) and External Quality (EQ). The Functional software features contain sub features of Data and Control Transfer (DCT), Functional Specification (FS), Behaviour (B), and Functionality (Fnl). The membership functions defined for these inputs are low, medium and high. While the membership functions for output are very low, low, medium, high and very high. Figure 5.2 shows the generic representation of the process of fuzzy logic.

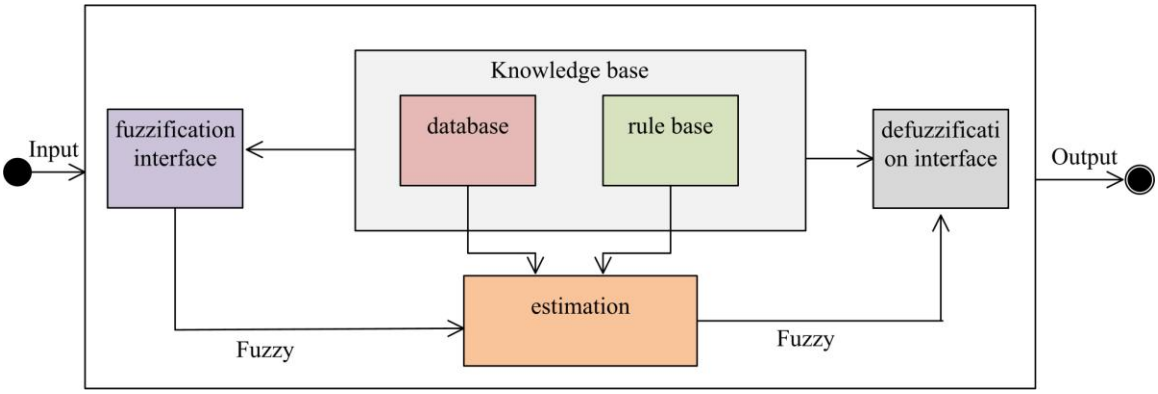

Figure 5.2. Generic view of the fuzzy logic process

The proposed methodology for estimation is carried out by using Matlab fuzzy tool box. Figure 5.3 shows the process of the proposed fuzzy model for estimation of software features based birthmark.

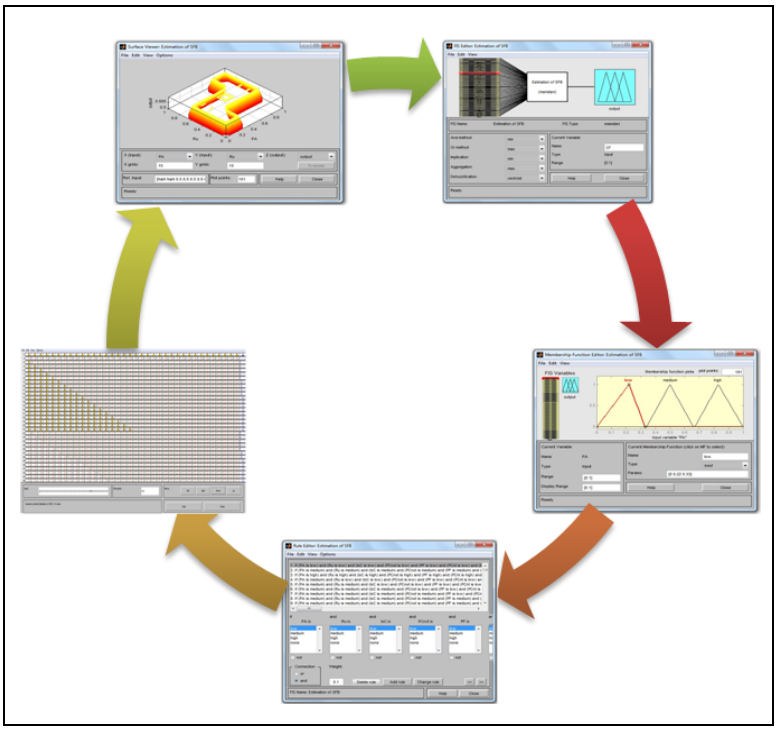

Figure 5.3. Process of the proposed fuzzy model for estimation of software features

5.4 Rules for estimation of software features

The proposed method for estimation of software features based birthmark is based on fuzzy rules. These rules were designed on the basis of membership functions. X. Xie et al. [48] pointed out that the evaluation measure be put forwarded to estimate the effectiveness of birthmark. S. Choi et al. [31] presents that API call sequence can be estimated using flow graphs. The dynamic characteristics of a program can be estimated through the control flow edge [47]. T. Kalker et al. [77] estimated watermark through detector analysis. The rules for estimation of software features based birthmark are in the form as below.

R 1. If (PA is low) and (Ru is low) and (IoC is low) and (PCnxt is low) and (PF is low) and (PCnt is low) and (IDS is low) and (PR is low) and (CT is low) and

(CF is low) and (NoSP is low) and (Na is low) and (F is low) and (ID is low) and (RLC is low) and (SoP is low) and (CD is low) and (GDS is low) and (UI is low) and (IQ is low) and (A is low) and (EoU is low) and (UF is low) and (Sc is low) and (Ap is low) and (IC is low) and (R is low) and (D is low) and (P is low) and (S is low) and (Std is low) and (EQ is low) and (DCT is low) and (FS is low) and (B is low) and (Fnl is low) then (output is very_low) (0.1)

R 2. If (PA is medium) and (Ru is medium) and (IoC is medium) and (PCnxt is medium) and (PF is medium) and (PCnt is medium) and (IDS is medium) and (PR is medium) and (CT is medium) and (CF is medium) and (NoSP is medium) and (Na is medium) and (F is medium) and (ID is medium) and (RLC is medium) and (SoP is medium) and (CD is medium) and (GDS is medium) and (UI is medium) and (IQ is medium) and (A is medium) and (EoU is medium) and (UF is medium) and (Sc is medium) and (Ap is medium) and (IC is medium) and (R is medium) and (D is medium) and (P is medium) and (S is medium) and (Std is medium) and (EQ is medium) and (DCT is medium) and (FS is medium) and (B is medium) and (Fnl is medium) then (output is medium) (0.5)

R 3. If (PA is high) and (Ru is high) and (IoC is high) and (PCnxt is high) and (PF is high) and (PCnt is high) and (IDS is high) and (PR is high) and (CT is high) and (CF is high) and (NoSP is high) and (Na is high) and (F is high) and (ID is high) and (RLC is high) and (SoP is high) and (CD is high) and (GDS is high) and (UI is high) and (IQ is high) and (A is high) and (EoU is high) and (UF is high) and (Sc is high) and (Ap is high) and (IC is high) and (R is high) and (D is high) and (P is high) and (S is high) and (Std is high) and (EQ is high) and (DCT

is high) and (FS is high) and (B is high) and (Fnl is high) then (output is very_high) (1).

Figure 5.4 shows the nomenclature of inputs, membership function and output generated in the Matlab.

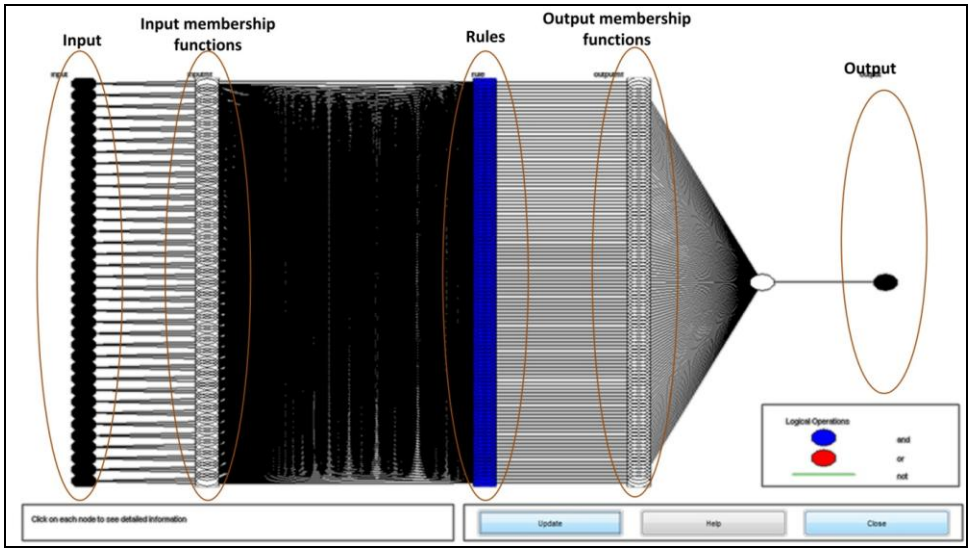

Figure 5.4. Nomenclature of the inputs, membership function and output

Figure 5.5 visually shows the proposed method for estimation of software features based birthmark.

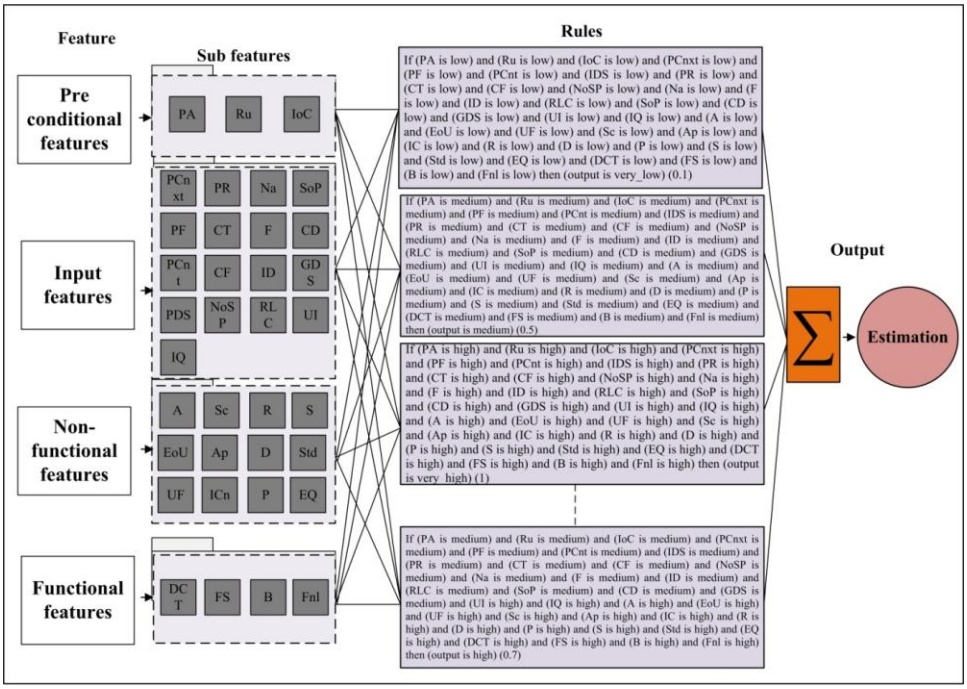

Figure 5.5. Proposed method for the estimation of software features based birthmark

Different rules are obtained based on the membership functions. The rule view is used to show the entire process of the inference system from start to end. A "ruleviewer" is displayed through command ruleview ('a') and show the fuzzy inference diagram for a FIS (a). Figure 5.6 shows the structure of rules viewer.

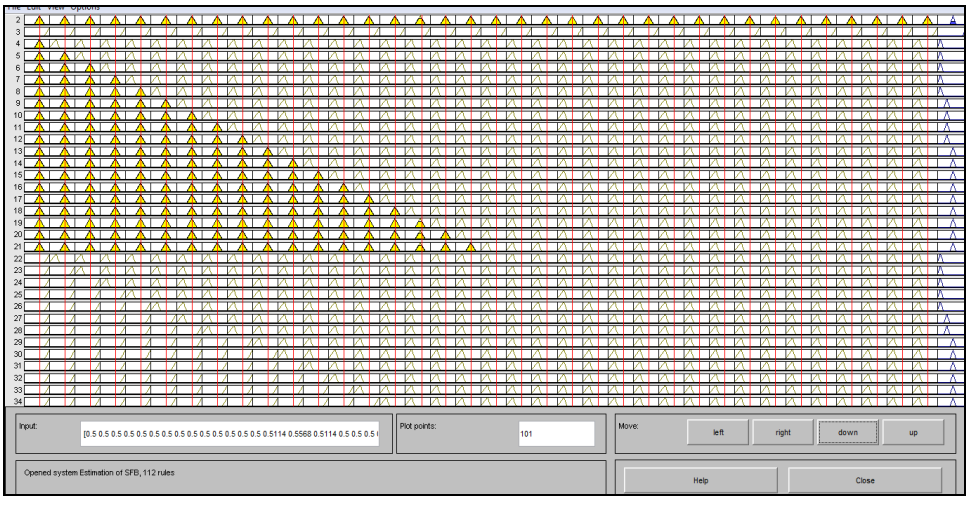

Figure 5.6. Rules viewer

The surface viewer is a read only graphical user interface tool which shows the output surface of a fuzzy inference system stored in a database file (name ‘a’) for inputs. It can be displayed through command surfview (‘a’). With the help of dropdown menus, the two inputs can be selected according to own choice for the input axis (X and Y) and for output axis (Z). Dragging the mouse and clicking on the plot axes, the surface can be manipulated so that it can be viewed in different angle shapes.

Surface viewer of the inputs Ru and PA is shown in figure 5.7. The other surface viewer (for other features) can also be plotted in the same way.

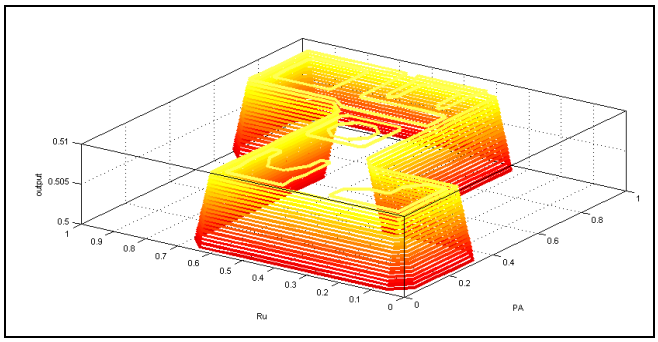

Figure 5.7. Surface viewer (Ru and PA)

Surface viewer of the inputs IoC and PA is shown in figure 5.8.

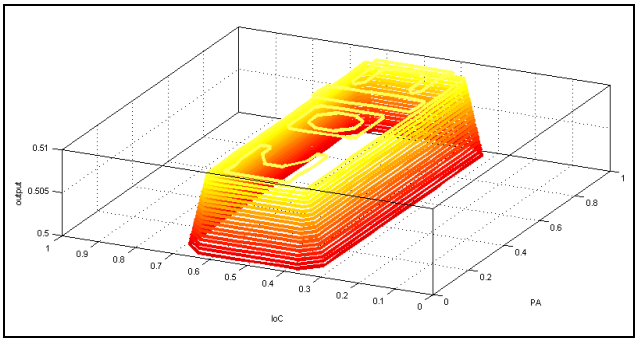

Figure 5.8. Surface viewer (IoC and PA)

5.5 Derivation process for weight consensus of software birthmark

The weights for various features of the software are results of a brainstorming and critical thinking among several domain experts. These weights are the qualitative or quantitative number assigned to different features of the software. Many researchers have reported the success of this process in gathering critical information regarding important aspects of elements of diverse nature [98-100]. In the context of this research the selected experts have been actively involved in the development of software solution for universities. The experts reviewed various features of the software under study, including precondition features, input features, functional features and nonfunctional features. The outcome of the process of such a conscious and critical thinking is a set of weights for all the features. Each weight value assigned to an individual feature defines the strength or weakness of the particular feature in the context of the software under study.

In the proposed study initially a group of ten experts who started with a detailed review of the domain and design of the software and the feature based birthmark. The experts were provided with potentially four different copies of presumably the same software. The experts came up with the relative weights of all features of the software(s) after a long discussion sessions. There were variations in opinions of different experts in which extreme values were excluded and the average of weights was taken. This resulted in the relative consensus weight of each feature. After deciding over the weights the proposed estimation process was performed through the designed model. Figure 5.9 shows the process of deriving weights from experts in the present study.

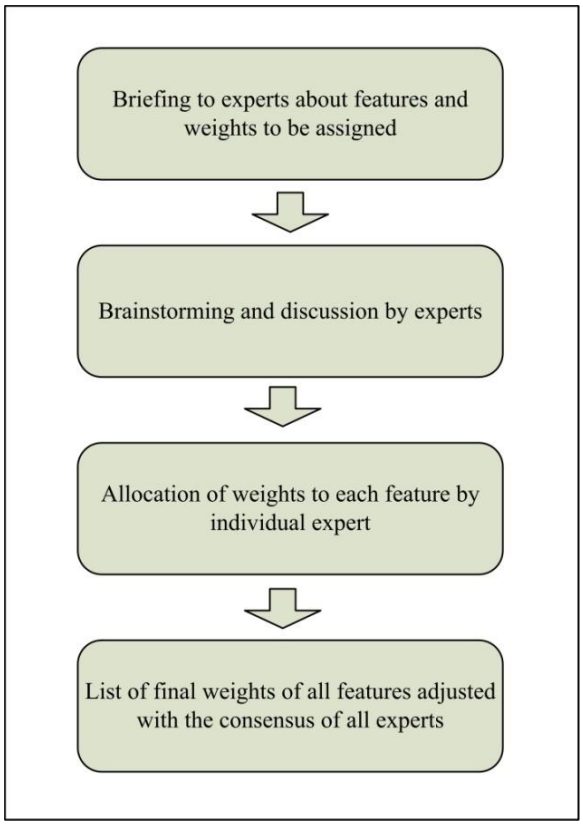

Figure 5.9. Protocol for experts meeting for weight assignment

5.6 Results and discussion

A fuzzy model has been designed which estimates the software features based birthmark for credibility and resilience. Input values of the features were derived from experts and passed through the proposed system. The structure of the proposed system, inputs and output is shown in table 5.1;

Table 5.1. Structure of the proposed system (inputs and output)

System	Inputs	Output
Name='Estimation of Software features based birthmark' Type='mamdani' Version=2.0 NumInputs=36 NumOutputs=1 AndMethod='min' OrMethod='max' ImpMethod='min' AggMethod='max' DefuzzMethod='centroid'	Name='PA' Range=[0 1] NumMFs=3 MF1='low':'trimf',[0 0.22 0.33] MF2='medium':'trimf',[0.34 0.5 0.66] MF3='high':'trimf',[0.67 0.83 1] . . . . [Input36] Name='Fnl' Range=[0 1] NumMFs=3 MF1='low':'trimf',[0 0.22 0.33] MF2='medium':'trimf',[0.34 0.5 0.66] MF3='high':'trimf',[0.67 0.83 1]	Name='output' Range=[0 1] NumMFs=5 MF1='very_low':'trimf',[0 0.1 0.2] MF2='medium':'trimf',[0.414 0.509259259259259 0.605] MF3='high':'trimf',[0.612 0.715608465608466 0.829] MF4='low':'trimf',[0.21 0.31 0.41] MF5='very_high':'trimf',[0.835 0.908730158730159 1]

The sequence of inputs in a specific format was given to the model according to the features of the software's and is shown in table 5.2.

Table 5.2. Inputs and output

No.	Inputs (PA, Ru, IoC, PCnxt, PF, PCnt, IDS, PR, CT, CF, NoSP, Na, F, ID, RLC, , SoP, CD, GDS, UI, IQ, A, EoU, UF, Sc, Ap, IC, R, D, P, S, Std, EQ, DCT, FS, B and Fnl)	Output
1	out = evalfis([0.7 0.7 0.7 0.7 0.8 0.7 0.8 0.9 0.8 0.9 0.7 0.9 0.7 0.8 0.7 0.8 0.9 0.8 0.9 0.7 0.8 0.9 0.9 0.8 0.7 0.8 0.9 0.9 0.9 0.8 0.8 0.8 0.9 0.8 0.9 0.9], fismat)	0.91
2	out = evalfis([0.5 0.5 0.6 0.7 0.8 0.7 0.6 0.6 0.8 0.9 0.1 0.3 0.2 0.3 0.2 0.1 0.2 0.5 0.6 0.1 0.3 0.2 0.3 0.2 0.7 0.2 0.2 0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.4 0.9], fismat)	0.50
3	out = evalfis([0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.2 0.2 0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.2 0.2 0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.2 0.2 0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.2 0.2], fismat)	0.10
4	out = evalfis([0.5 0.5 0.6 0.7 0.8 0.7 0.6 0.6 0.8 0.9 0.7 0.6 0.7 0.6 0.7 0.8 0.6 0.5 0.6 0.7 0.6 0.5 0.6 0.8 0.7 0.2 0.2 0.1 0.1 0.3 0.2 0.3 0.2 0.1 0.4 0.9], fismat)	0.50

So, from the output of table 5.2 the extent of piracy can be checked, that is how much software is pirated in term of the given features.

Summary

This chapter includes the details of identifying different software features forming as a birthmark. Four different software features are identified that are preconditioned, input, functional and nonfunctional software features. These features are further sub categorized into 36 different features. Furthermore, these features forming as a birthmark is estimated through fuzzy logic.

Chapter 6

Mathematical modelling for detection of software piracy

Software birthmark has been discussed in the previous chapters and it has been identified by this research that use of feature based software birthmark and the process of estimating software birthmark can together provide an appropriate and powerful technique to detect software piracy and extent of piracy performed in software. There is also a need to have an objective measure to compare birthmarks of software(s) to detect pirated software. Software development industry has been employing different techniques for the detection and identification of software theft. These techniques mainly include advanced versions of software watermarks and fingerprints [1-14]. All these techniques used in software industry for the said purpose have some limitations (explained in chapter 2), due to which these technique have now become less popular. This chapter discusses a mathematical model which can be used for detection of software piracy in terms of birthmark. The method discussed in this chapter is based on the concept of feature based software birthmark, discussed in chapter 4 of this thesis [101].

6.1 Need for a mathematical model

The comparison of suggested feature based software birthmark [101] is mathematically modeled to facilitate the comparison of birthmark on the basis of the defined features. This feature based comparison suggests the similarity among different software(s).

Different mathematical techniques are used by researchers for modeling different real life phenomena. Such techniques include separable variable methods, linear equations, exact

equations, solution by substitution and numerical methods. These methods are used for solving first order differential equations [102]. In the proposed research work we can design the required model in the form of homogeneous linear differential system. For such kind of systems three methods are commonly used, named as distinct real Eigen values, repeated Eigen values and complex Eigen values. In the context of this research Eigen values are complex.

Mathematically, if $\lambda_1 = \alpha + i\beta$ and $\lambda_2 = \alpha - i\beta$, where i^2= -1 are complex Eigen values of the matrix "A". Then the corresponding Eigen vector has also complex entries [102].

6.2 Terminologies used for modelling software piracy detection

The following sections discuss the techniques and terminologies used for modelling the detection of software piracy.

6.2.1 Differential model for birthmark

The differential equations have the derivatives of one or more dependent variable(s), with respect to one or more independent variable(s) [102]. Suppose we have an equation, and we don't know how it was constructed. What the function represented by the symbol(s). For example how we solve an equation of unknown function y'= ϕ(x)?

6.2.2 Eigen values and Eigenvector

The characteristic polynomial of a square matrix "A" is defined by [103]:

$$p(\lambda) = \det(A - \lambda I) \qquad (6.1)$$

If p is the characteristic polynomial of matrix “A”, then the roots of p are the Eigen values of matrix “A”. If λ is Eigen value of “A” and $x \neq 0$ satisfies $(A-\lambda I)x = 0$. Then x is Eigen vector of a corresponding to the Eigen value λ.

6.3 The model for detection of software piracy

The proposed method for comparison of suggested feature based software birthmark is mathematically modeled for facilitation of the comparison of birthmark on the basis of the defined features. These features are already identified in the previous work [101]. This feature based comparison suggests the similarity among different software(s). Here, we considered the four main features that were identified in [101]. These features include pre conditional features, input features, nonfunctional features and functional features. The category of pre conditional features has further three sub features that are program availability, runnable and identification of components. These are the important features which can be checked initially for every program that is to be checked for similarity. Figure 6.1 shows the detail of feature based software birthmark, as defined in chapter 4 [101].

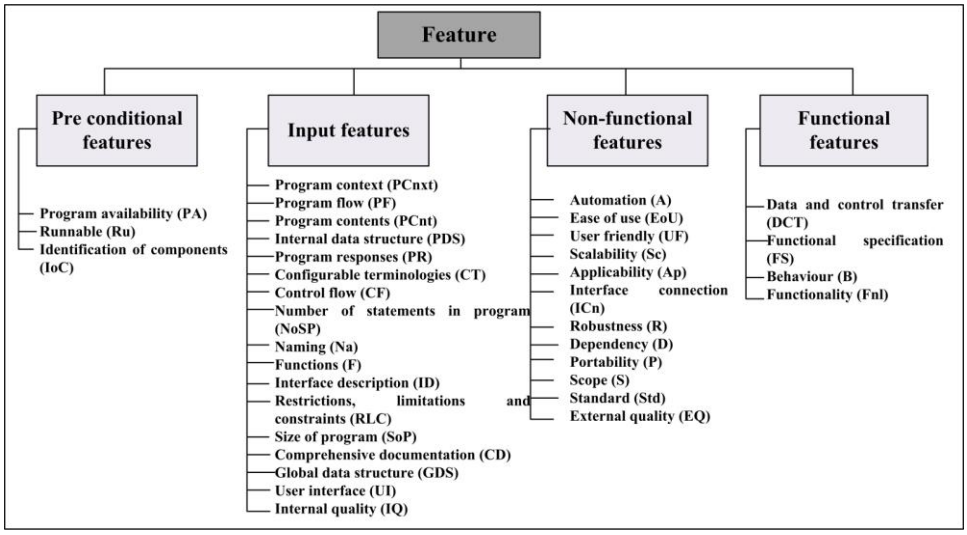

Figure 6.1. Software features

After doing this initial analysis rest of the three features categories are then used for mathematical modeling. The input feature category has further 17 features that are program context, program flow, program contents, internal data structure, program responses, configurable terminologies, control flow, number of statements in program, naming, functions, interface description, restriction, limitation and constraints, size of program, comprehensive documentation, global data structure, user interface, and internal quality. The nonfunctional feature include 12 sub features that are automation, ease of use, friendly, scalability, applicability, interface connections, robustness, dependency, portability, scope, standard, and external quality. The functional feature is having further four sub features that are data & control process, functional specification, behavior, and functionality. These features can be plotted mathematically in the form of differential system as;

$$\begin{pmatrix} x'(f) = 17x + 12y + 4z \\ y'(f) = 4x + 17y + 12z \\ z'(f) = 12x + 4y + 17z \end{pmatrix} \qquad (6.2)$$

Where x, y and z are the three features.

The matrix form of (5.2) is;

$$\begin{pmatrix} x'(f) \\ y'(f) \\ z'(f) \end{pmatrix} = \begin{pmatrix} 17 & 12 & 4 \\ 4 & 17 & 12 \\ 12 & 4 & 17 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$X'(f) \qquad A \qquad X$$

$$X'(f) = A\,X(f) \qquad (6.3)$$

To find these three features x, y and z, we need to find the solution of the system (6.3). For this purpose we find the Eigen values of the matrix A and Eigen vector corresponding to these Eigen values. The proposed process has been carried in the following steps.

Step 1. To find Eigen value

$$\text{Since} \quad A = \begin{bmatrix} 17 & 12 & 4 \\ 4 & 17 & 12 \\ 12 & 4 & 17 \end{bmatrix}$$

According to section 6.2.2, by using equation (6.1), the characteristic polynomial of the matrix A is given by $\det(A-\lambda I) = 0$.

$$\det\left(\begin{bmatrix} 17 & 12 & 4 \\ 4 & 17 & 12 \\ 12 & 4 & 17 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) = 0$$

$$\det\left(\begin{bmatrix} 17 & 12 & 4 \\ 4 & 17 & 12 \\ 12 & 4 & 17 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}\right) = 0$$

$$\det\begin{bmatrix} 17-\lambda & 12-0 & 4-0 \\ 4-0 & 17-\lambda & 12-0 \\ 12-0 & 4-0 & 17-\lambda \end{bmatrix} = 0$$

$$\text{i.e. } \begin{vmatrix} 17-\lambda & 12-0 & 4-0 \\ 4-0 & 17-\lambda & 12-0 \\ 12-0 & 4-0 & 17-\lambda \end{vmatrix} = 0$$

Expanding by first row, we have

$$(17-\lambda)\begin{vmatrix} 17-\lambda & 12 \\ 4 & 17-\lambda \end{vmatrix} - 12\begin{vmatrix} 4 & 12 \\ 12 & 17-\lambda \end{vmatrix} + 4\begin{vmatrix} 4 & 17-\lambda \\ 12 & 4 \end{vmatrix} = 0$$

$$(17-\lambda)\left((17-\lambda)^2-48\right) - 12\left(68-4\lambda-144\right) + 4\left(16-12(17-\lambda)\right) = 0$$

$$(17-\lambda)\left(289+\lambda^2-34\lambda-48\right) - 12\left(-4\lambda-76\right) + 4\left(12\lambda-188\right) = 0$$

$$4913+17\lambda^2-578\lambda-816-289\lambda-\lambda^3+34\lambda^2+48\lambda+48\lambda+912+48\lambda-752=0$$

$$-\lambda^3+51\lambda^2-723\lambda+4257=0$$

$$\lambda^3-51\lambda^2+723\lambda-4257=0$$

By using syntactic division, we have

$$\lambda_1 = 33$$
$$\lambda_2 = 9+6.9282\text{i}$$
$$\lambda_3 = 9-6.9282\text{i}$$

Thus the Eigen values of the matrix A are 33, 9+6.9282i and 9-6.9282i. Where λ_1 is real, λ_2 is complex and λ_3 is complex conjugate of λ_2.

Step 2. To find Eigen vectors of corresponding Eigen values

If λ=33, then the corresponding Eigen vector is given by AX=λX.

$$\begin{bmatrix} 17 & 12 & 4 \\ 4 & 17 & 12 \\ 12 & 4 & 17 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = 33\begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

By solving this equation we get

$$-16a+12b+4c=0$$

$$4a-16b+12c=0$$

$$12a+4b-16c=0$$

By solving this we have $a=1, b=1, \text{and } c=1$

Thus the Eigen vectors corresponding to $\lambda_1, \lambda_2, \text{and } \lambda_3$ are $V1=\begin{pmatrix}1\\1\\1\end{pmatrix}$

$$V_2=\begin{pmatrix}\frac{1}{2}i\left(i+\sqrt{3}\right)\\-\frac{1}{2}i\left(-i+\sqrt{3}\right)\\1\end{pmatrix} \qquad \text{And} \qquad V_3=\begin{pmatrix}-\frac{1}{2}i\left(-i+\sqrt{3}\right)\\\frac{1}{2}i\left(i+\sqrt{3}\right)\\1\end{pmatrix}$$

Step 3. Thus the solution of system (6.3) is given by

$$X=c_1V_1e^{\lambda_1 f}+c_2\left(B_1\cos\beta f-B_2\sin\beta f\right)e^{\alpha f}+c_3\left(B_2\cos\beta f+B_1\sin\beta f\right)e^{\alpha f}$$

Where $\lambda=\alpha+i\beta$, B_1 =Real part (Eigen vector) and B_2 = Imaginary part (Eigen vector).

Putting the values in the above equation, we get

$$X=c_1\begin{pmatrix}1\\1\\1\end{pmatrix}e^{33f}+c_2\left(\begin{pmatrix}-\frac{1}{2}\\-\frac{1}{2}\\1\end{pmatrix}\cos 6.9282f-\begin{pmatrix}\frac{\sqrt{3}}{2}\\-\frac{\sqrt{3}}{2}\\0\end{pmatrix}-\sin 6.9282f\right)e^{9f}+c_3\left(\begin{pmatrix}\frac{\sqrt{3}}{2}\\-\frac{\sqrt{3}}{2}\\0\end{pmatrix}\cos 6.9282f+\begin{pmatrix}-\frac{1}{2}\\-\frac{1}{2}\\1\end{pmatrix}\sin 6.9282f\right)e^{9f}$$

$$x(f)=c_1e^{33f}+c_2\left(\cos\left(6.9282f\right)\right)e^{9f}+c_3\left(\sin\left(6.9282f\right)\right)e^{4f}$$

$$y(f)=c_1e^{33f}+c_2\left[-\frac{1}{2}\cos\left(6.9282f\right)+\frac{\sqrt{3}}{2}\sin\left(6.9282f\right)\right]e^{9f}+c_3\left[-\frac{\sqrt{3}}{2}\cos\left(6.9282f\right)-\frac{1}{2}\sin\left(6.9282f\right)\right]e^{9f}$$

$$z(f) = c_1 e^{33f} + c_2\left[-\frac{1}{2}\cos(6.9282f) - \frac{\sqrt{3}}{2}\sin(6.9282f)\right]e^{9f} + c_3\left[\frac{\sqrt{3}}{2}\cos(6.9282f) - \frac{1}{2}\sin(6.9282f)\right]e^{9f}$$

Put the value of f=0 in the above equations and using initial conditions, we have

$$c_1 - \frac{1}{2}c_2 + \frac{\sqrt{3}}{2}c_3 = 17$$
$$c_1 - \frac{1}{2}c_2 - \frac{\sqrt{3}}{2}c_3 = 4$$
$$c_1 + c_2 = 12$$

By solving these equations, we get

$$c_1=11,\ c_2=1 \text{ and } c_3=-7.5056$$

$$x(f) = 11e^{33f} + \cos(6.9282f)e^{9f} - 7.5056\sin(6.9282f)e^{4f}$$

$$y(f) = 11e^{33f} + \left[-\frac{1}{2}\cos(6.9282f) + \frac{\sqrt{3}}{2}\sin(6.9282f)\right]e^{9f} - 7.5056\left[-\frac{\sqrt{3}}{2}\cos(6.9282f) - \frac{1}{2}\sin(6.9282f)\right]e^{9f}$$

$$z(f) = 11e^{33f} + \left[-\frac{1}{2}\cos(6.9282f) - \frac{\sqrt{3}}{2}\sin(6.9282f)\right]e^{9f} - 7.5056\left[\frac{\sqrt{3}}{2}\cos(6.9282f) - \frac{1}{2}\sin(6.9282f)\right]e^{9f}$$

This is the required solution of the differential system (6.2) for the available features of software birthmark.

In order to compare software birthmarks, multiple instances of software defined over same feature based birthmark [101] can be modeled using the differential system defined in the previous sections. If the solution of the resulting differential systems is same or close to same, then the software(s) are copy of each other, hence pirated.

Summary

This chapter presents a mathematical model for software piracy detection process. Three features categories of software are considered under the proposed study. These features include input feature, nonfunctional feature and functional feature. These features are further categorized in the form of differential system. Exact solution of these features has to be produced. This solution can be then be compared with the solution obtained from the pirated copy of the software to show if the software is pirated.

Chapter 7

Conclusion and future work

Software piracy has turned out to be a major concern due to the extravagant development of software industry and the Internet. Broad research into techniques of software piracy detection has prompted development of techniques like software watermarking, finger prints, and lately the software birthmarks. With the development of advanced techniques and countermeasures such as code obfuscation, software optimization, and semantic transformations, use of watermarking has become inadequate and illogical on extent. Whereas, the concept of software birthmark is successfully used in detecting software theft and piracy. Estimation of software birthmark can play a key role in accepting the effectiveness of a birthmark. In this research an estimation model based on fuzzy logic has been proposed. In the context of estimation of software birthmarks situations of uncertainty may arise. The proposed model of fuzzy rules works well in case of uncertainty and with unknown information. The model is based on the two properties of software birthmark; credibility and resilience. As the process is based on gathering expert opinion regarding software birthmark, therefore, the process can be used for different types of software birthmark(s). Results produced by the proposed process show that the method is efficient and provides satisfactory results. The approach has been tested for credibility and resilience, as these two properties are considered as most important properties of software birthmark(s). The second objective of this research is to define a more useful and technically efficient software birthmark. Software features represent many unique properties of software; hence a collection of certain features can act as

birthmark for the software. This birthmark can then be used for several purposes, most importantly for software piracy detection or theft detection. Features of two different programs may be compared to check if the software programs are copy of each other.

The research also proposes the estimation model for the proposed feature based software birthmark. The model, again, estimates the birthmark in term of credibility and resilience. It accepts input values for 36 different software features on which the birthmark is based. These input values are processed by the model using predefined fuzzy membership functions and rules. The results of the study clearly show the validity of the proposed method, and hence, give efficient results in term of specified features.

Finally, a mathematical model has been presented to compare software birthmarks. The proposed feature based birthmark along with the estimation process and the mathematical model can prove to be a comprehensive technique to tackle software piracy and theft.

7.1 Future work and limitations

The proposed features based birthmark model may be extended to add more features into the defined categories. Also, identification of specific techniques (such as qualitative and quantitative) to detect software features is a task to further formalize the model. There should be an extension of model, to store important information about different features (birthmark) of the software. This information will help researchers in identifying highly pirated software(s) and will also be used as evidence against pirates.

References

[1] G. Myles and C. Collberg, "Software Watermarking Through Register Allocation: Implementation, Analysis, and Attacks," in *Information Security and Cryptology - ICISC 2003*. vol. 2971: Springer Berlin Heidelberg, pp. 274-293, 2004.

[2] C. Collberg and T. R. Sahoo, "Software watermarking in the frequency domain: Implementation, analysis, and attacks," *Journal of Computer Security,* vol. 13, pp. 721–755, 2005.

[3] F. Liu, B. Lu, and X. Luo, "A Chaos-Based Robust Software Watermarking," in *Information Security Practice and Experience*. vol. 3903: Springer Berlin Heidelberg, pp. 355-366, 2006.

[4] H. Park, S. Choi, H.-i. Lim, and T. Han, "Detecting code theft via a static instruction trace birthmark for Java methods," in *6th IEEE International Conference on Industrial Informatics*, pp. 551-556, 2008.

[5] H. Park, S. Choi, H.-i. Lim, and T. Han, "Detecting Java Theft Based on Static API Trace Birthmark," in *Advances in Information and Computer Security*. vol. 5312: Springer Berlin Heidelberg, pp. 121-135, 2008.

[6] H.-i. Lim, H. Park, S. Choi, and T. Han, "A method for detecting the theft of Java programs through analysis of the control flow information," *Information and Software Technology*, vol. 51, pp. 1338–1350, 2009.

[7] Y. Zeng, F. Liu, X. Luo, and C. Yang, "Software Watermarking Through Obfuscated Interpretation: Implementation and Analysis," *Journal of Multimedia,* vol. 6, pp. 329-340, 2011.

[8] H. Park, H.-i. Lim, S. Choi, and T. Han, "Detecting common modules in Java packages based on static object trace birthmark," *Computer Journal*, vol. 54, pp. 108-124, 2011.

[9] G. e. Arboit, "A method for watermarking java programs via opaque predicates," in *The 5th International Conference on Electronic Commerce Research (ICECR-5)*, pp. 1-8, 2002.

[10] C. Collberg, E. Carter, S. Debray, A. Huntwork, C. Linn, and M. Stepp, "Dynamic path-based software watermarking," in *In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 04)*, pp. 1-10, 2004.

[11] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in *Conference Record of POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Jan.1999)*, http://citeseer.nj.nec.com/collberg99software.html., pp. 311-324, 1999.

[12] A. Monden, H. Iida, K.-i. Matsumoto, K. Inoue, and K. Torii, "A practical method for watermarking java programs," in *24th Computer Software and Applications Conference*, pp. 191-197, 2000.

[13] J. P. Stern, G. e. Hachez, F. c. Koeune, and J.-J. Quisquater, "Robust Object Watermarking: Application to Code," in *Information Hiding*. vol. 1768: Springer Berlin Heidelberg, pp. 368-378, 2000.

[14] R. Venkatesan, V. Vazirani, and S. Sinha, "A Graph Theoretic Approach to Software Watermarking," in *4th International Information Hiding Workshop, Pittsburgh, PA*, pp. 157–168, 2001.

[15] A. Aiken, "Moss: A system for detecting software plagiarism," University of California–Berkeley. http://www. cs. berkeley. edu/aiken/moss. html, 2005.

[16] H. Tamada, M. Nakamura, and A. Monden, "Design and evaluation of birthmarks for detecting theft of Java programs," in *Proceedings of IASTED International Conference on Software Engineering*, pp. 569-575, 2004.

[17] Y. Guo, M. Wang, and Y. Luo, "Identifying Software Theft Based on Classification of Multi-Attribute Features," *Journal of Software*, vol. 9, pp. 1401-1411, 2014.

[18] S. Cesare and Y. Xiang, "*Software Similarity and Classification"*. New York Dordrecht: Springer London Heidelberg, 2012.

[19] BSA, "The Compliance Gap BSA Global Software Survey," Business Software Alliance, 2014.

[20] G. Myles and C. Collberg, "Detecting Software Theft via Whole Program Path Birthmarks," in *Information Security*. vol. 3225: Springer Berlin Heidelberg, pp. 404-415, 2004.

[21] R. Thabit and B. E. Khoo, "Robust reversible watermarking scheme using Slantlet transform matrix," *Journal of Systems and Software*, vol. 88, pp. 74-86, 2014.

[22] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 98. Digest of Technical Papers*, pp. 190-193, 1998.

[23] J. Pieprzyk, "Fingerprints for Copyright Software Protection," in *Information Security*. vol. 1729: Springer Berlin Heidelberg, pp. 178-190, 1999.

[24] C. S. Collberg, C. Thomborson, and G. M. Townsend, "Dynamic graph-based software fingerprinting," *ACM Trans. Program. Lang. Syst.,* vol. 29, pp. 35, 2007.

[25] H. Tamada. M. Nakamura, Monden, K. Matsumoto "Detecting the theft programs using birthmarks," Graduate School of Information Science, Nara Institute of Science and Technology, Japan, November 2003.

[26] H.-i. Lim, "Customizing k-Gram Based Birthmark through Partial Matching in Detecting Software Thefts," in *IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 1-4, 2013.

[27] Z. Xin, H. Chen, X. Wang, P. Liu, S. Zhu, B. Mao, and L. Xie, "Replacement attacks: automatically evading behavior-based software birthmark," *International Journal of Information Security,* vol. 11, pp. 293-304, 2012.

[28] H. Park, H.-i. Lim, S. Choi, and T. Han, "Detecting common modules in Java packages based on static object trace birthmark," *Computer Journal*, vol. 54, pp. 108-124, 2011.

[29] P. P. F. Chan, L. C. K. Hui, and S. M. Yiu, "Dynamic Software Birthmark for Java Based on Heap Memory Analysis," in *Communications and Multimedia Security*. vol. 7025: Springer Berlin Heidelberg, pp. 94-107, 2011.

[30] Y. Mahmood, S. Sarwar, Z. Pervez, and H. F. Ahmed, "Method based static software birthmarks: A new approach to derogate software piracy," in *2nd International Conference on Computer, Control and Communication*, pp. 1-6, 2009.

[31] S. Choi, H. Park, H.-i. Lim, and T. Han, "A static API birthmark for Windows binary executables," *Journal of Systems and Software,* vol. 82, pp. 862-873, 2009.

[32] H.-i. Lim, H. Park, S. Choi, and T. Han, "Detecting Theft of Java Applications via a Static Birthmark Based on Weighted Stack Patterns," *IEICE - Trans. Inf. Syst.,* vol. E91-D, pp. 2323-2332, 2008.

[33] J. Yang, J. Wang, and D. Li, "Detecting the Theft of Natural Language Text Using Birthmark," in *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 1-4, 2006.

[34] T. Kakimoto, A. Monden, Y. Kamei, H. Tamada, M. Tsunoda, and K.-i. Matsumoto, "Using software birthmarks to identify similar classes and major functionalities," in *Proceedings of the international workshop on Mining software repositories* Shanghai, China: ACM, pp. 171-172, 2006.

[35] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology,* vol. 55, pp. 1165-1199, 2013.

[36] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," in *Proceedings of the International Conference on Software Maintenance*: IEEE Computer Society, pp. 1-10, 1998.

[37] G. Whale, "Identification of program similarity in large populations," *Computer*, vol. 33, pp. 140–146, 1990.

[38] M. J. Wise, "Detection of similarities in student programs: YAP'ing may be preferable to plague'ing," in *23rd SIGCSE Technical Symposium*, pp. 268–271, 1992.

[39] S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of SIGMOD Conference*, 2003.

[40] Z. Tian, Q. Zheng, T. Liu, M. Fan, X. Zhang, and Z. Yang, "Plagiarism detection for multithreaded software based on thread-aware software birthmarks," in *Proceedings of the 22nd International Conference on Program Comprehension* Hyderabad, India: ACM, pp. 304-313, 2014.

[41] Z. Tian, Q. Zheng, T. Liu, and M. Fan, "DKISB: Dynamic Key Instruction Sequence Birthmark for Software Plagiarism Detection," in *IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 619-627, 2013.

[42] Y. Zeng, F. Liu, X. Luo, and S. Lian, "Abstract interpretation-based semantic framework for software birthmark," *Computers & Security*, vol. 31, pp. 377-390, 2012.

[43] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Dynamic Software Birthmarks to Detect the Theft of Windows Applications," in *Int. Symp. on Future Software Technology*, pp. 1-6, 2004.

[44] Z. Xin, H. Chen, X. Wang, P. Liu, S. Zhu, B. Mao, and L. Xie, "Replacement Attacks on Behavior Based Software Birthmark," in *LNCS*, pp. 1-16, 2011.

[45] K. Fukuda and H. Tamada, "A Dynamic Birthmark from Analyzing Operand Stack Runtime Behavior to Detect Copied Software," in *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 505-510, 2013.

[46] Y. Bai, X. Sun, G. Sun, X. Deng, and X. Zhou, "Dynamic K-gram based Software Birthmark," in *19th Australian Conference on Software Engineering*, pp. 644-649, 2009.

[47] H.-i. Lim, H. Park, S. Choi, and T. Han, "A Static Java Birthmark Based on Control Flow Edges," in *23rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pp. 413-420, 2009.

[48] X. Xie, F. Liu, B. Lu, and L. Chen, "A Software Birthmark Based on Weighted K-gram," in *IEEE International Conference on Intelligent Computing and Intelligent System (ICIS)*, pp. 400-405, 2010.

[49] Y. Mahmood, Z. Pervez, S. Sarwar, and H. F. Ahmed, "Similarity Level Method Based Static Software Birthmarks," in *High Capacity Optical Networks and Enabling Technologies*, pp. 205-210, 2008.

[50] Y. Wang, F. Liu, Z. Zhao, B. Lu, and X. Xie, "Operand Stack Dependence Based Java Static Software Birthmark," in *10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* pp. 1090-1095, 2013.

[51] X. Zhou, X. Sun, G. Sun, and Y. Yang, "A Combined Static and Dynamic Software Birthmark Based on Component Dependence Graph," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 1416-1421, 2008.

[52] G. Sun, "Software Birthmark Based on Component Dependence Graph Cluster," in *International Conference on Computer Application and System Modeling (ICCASM 2010)*, pp. 281-291, 2010.

[53] J. Choi, Y. Han, S.-j. Cho, HaeYoungYoo, and J. Woo, "A Static Birthmark for MS Windows Applications Using Import Address Table," in *7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 129-134, 2013.

[54] L. Ma, Y. Wang, F. Liu, and L. Chen, "Instruction-Words Based Software Birthmark," in *4th International Conference on Multimedia Information Networking and Security (MINES)*, pp. 909-912, 2012.

[55] H. Kim, W. M. Khoo, and P. Li`o, "Polymorphic Attacks against Sequence-based Software Birthmarks," in *2nd Software Security and Protection Workshop (SSP'12)*, pp. 1-8, 2012.

[56] D. Lee, Y. Choi, J. Jung, J. Kim, and D. Won, "An Efficient Categorization of the Instructions Based on Binary Executables for Dynamic Software Birthmark," *International Journal of Information and Education Technology*, vol. 5, pp. 571-576, 2015.

[57] Y. Wang, F. Liu, D. Gong, B. Lu, and S. Ma, "CHI Based Instruction-Words Based Software Birthmark Selection," in *4th International Conference on Multimedia Information Networking and Security*, 2012, pp. 892-895.

[58] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1-161, 1990.

[59] J. Kalaoja, E. Niemela, and H. Perunka, "Feature Modelling of Component-Based Embedded Software," in *8th IEEE International Workshop on incorporating Computer Aided Software Engineering*, pp. 444-451, 1997.

[60] P. Wang, C. Jin, and S.-W. Jin, "Software Defect Prediction Scheme Based on Feature Selection," in 4th *International Symposium on Information Science and Engineering*, 2012, pp. 477-480.

[61] Y. Zheng, F. Liu, X. Luo, and C. Yang, "A Method Based on Feature Matching to Identify steganography software," in 4th *International Conference on Multimedia Information Networking and Security*, pp. 989-994, 2012.

[62] Y. He, "Tamperproofing a Software Watermark by Encoding Constants." Master of Science in Computer Science: University of Auckland, 2002, pp. 1-157.

[63] P. Cousot and R. Cousot, "An abstract interpretation-based framework for software watermarking," Symposium on Principles of Programming Languages, Venice, Italy, vol. 39, pp. 173-185, 2004.

[64] H. Park, H.-i. Lim, S. Choi, and T. Han, "Detecting common modules in Java packages based on static object trace birthmark," *Computer Journal*, vol. 54, pp. 108-124, 2009.

[65] Y. Bai, X. Sun, G. Sun, X. Deng, and X. Zhou, "Dynamic k-gram based software birthmark," in *IEEE ASWEC 2008 19th Australian Conference*, pp. 644-649, 2008.

[66] G. Myles and C. Collberg, "K-gram based software birthmarks," in *Proceedings of the ACM symposium on Applied computing* Santa Fe, New Mexico: ACM , pp. 314-318, 2005.

[67] X. Wang, "Protecting Software from Attacks and Theft via Program Analysis." Doctor of Philosophy: The Pennsylvania State University The Graduate School, 2009.

[68] G. M. Myles, "Software theft detection through program identification," in *Department of Computer Science*. Doctor of Philosophy: The University of Arizona, pp. 1-351, 2006.

[69] H. Tamada, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Design and evaluation of birthmarks for detecting theft of java programs," in *IASTED International Conference on Software Engineering (IASTED SE 2004)*, pp. 569–575, 2004.

[70] K. Lin, L. Yuan, and G. Qu, "SecureGo: A Hardware-Software Co-Protection against Identity Theft in Online Transaction," in *Bio-inspired, Learning, and Intelligent Systems for Security. BLISS 2007. ECSIS Symposium on*, pp. 59-64, 2007.

[71] S. Mumtaz, S. Iqbal, and I. Hameed, "Development of a Methodology for Piracy Protection of Software Installations," in *9th International Multitopic Conference, IEEE INMIC 2005*, pp. 1-7, 2005.

[72] C. Christian, M. Ginger, and H. Andrew, "Sandmark--A Tool for Software Protection Research," *IEEE Security and Privacy*, vol. 1, pp. 40-49, 2003.

[73] D. Curtis, "Software piracy and copyright protection," in *Wescon/94: Idea/Microelectronics* New York, NY, USA, pp. 199 - 203, 1994.

[74] F. Yaghmaee and M. Jamzad, "Estimating watermarking capacity in gray scale images based on image complexity," *EURASIP J. Adv. Signal Process,* vol. 2010, pp. 1-9, 2010.

[75] G.-R. Feng, L.-G. Jiang, D.-J. Wang, and C. He, "Quickly tracing detection for spread spectrum watermark based on effect estimation of the affine transform," *Pattern Recognition,* vol. 38, pp. 2530-2536, 2005.

[76] S. Voloshynovskiy, S. Pereira, A. Herrigel, N. Baumgartner, T. Pun "Generalized watermarking attack based on watermark estimation and perceptual remodulation," IS&,T/SPIE',s 12th Annual Symp., Electronic Imaging: Security and Watermarking of Multimedia Content II, SPIE Proc., vol. 3971, pp.358 - 370, 2000.

[77] T. Kalker, J.-P. Linnartz, and M. v. Dijk, "Watermark Estimation Through Detector Analysis " in *proceedings of the ICIP*, pp. 425-429, 1998.

[78] L. Zadeh, "Fuzzy Logic," *Computer,* vol. 1, pp. 83-93, 1988.

[79] *Fuzzy Logic Tool box ™ 2 User's Guide*: The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098, 1995–2010.

[80] Y.-J. W. M. Wasif Nisar, Manzoor Elahi, "Software Development Effort Estimation Using Fuzzy Logic - A Survey," 5th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 421-427, 2008.

[81] D. Ramot, M. Friedman, G. Langholz, and A. Kandel, "Complex Fuzzy Logic," *IEEE Transactions on Fuzzy Systems,* vol. 11, pp. 450-461, 2003.

[82] K. Seth, A. Sharma, and A. Seth, "Component Selection Efforts Estimation–a Fuzzy Logic Based Approach," *International Journal of Computer Science and Security (IJCSS)*, vol. 3, pp. 210-215, 2009.

[83] K. Tyagi and A. Sharma, "A rule-based approach for estimating the reliability of component-based systems," *Advances in Engineering Software*, vol. 54, pp. 24-29, 2012.

[84] "MATLAB," 7.10.0 ed Natick, Massachusetts: The MathWorks Inc, 2010.

[85] C. P. Ltd, "CodeShield Java Byte Obfuscator." vol. 2014 http://www.codingart.com/codeshield.html.

[86] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Behavior based software theft detection," in *Proceedings of the 16th ACM conference on Computer and communications security* Chicago, Illinois, USA: ACM, 2009.

[87] P. P. F. Chan, L. C. K. Hui, and S. M. Yiu, "Heap Graph Based Software Theft Detection," *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 101-110, 2013.

[88] A. Z. Broder, "On the resemblance and containment of documents," *Compression and Complexity of Sequences (SEQUENCES '97)*, pp. 21-29, 1998.

[89] C. S. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and ObfuscationÐTools for Software Protection," *IEEE Transactions on Software Engineering*, vol. 28, pp. 735-746, 2002.

[90] J. G. Shanthikumar, "On a software availability model with imperfect maintenance," *Operations Research Letter*, vol. 2, pp. 285-290, 1984.

[91] G. Caldiera and V. R. Basili, "Identifying and qualifying reusable software components," *Computer*, vol. 24, pp. 61-70, 1991.

[92] D. Birkmeier and S. Overhage, "On Component Identification Approaches – Classification, State of the Art, and Comparison," in *Component-Based Software Engineering*, pp. 1-18, 2009.

[93] S. M. H. Hasheminejad and S. Jalili, "SCI-GA: Software Component Identification using Genetic Algorithm," *Journal of Object Technology*, vol. 12, pp. 1-34, 2013.

[94] C. Prehofer, "Feature-Oriented Programming: A Fresh Look at Objects," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pp. 419-443, 1997.

[95] Doktoringenieur, "Measuring and Predicting Non-Functional Properties of Customizable Programs," Dissertation, Otto-von-Guericke-Universitat Magdeburg, Germany, 2012.

[96] L. M. Laird and M. C. Brennan, *Software Measurement and Estimation- A Practical Approach*: IEEE Computer Society, A John Wiley & Sons, Inc., Publication, 2006.

[97] J. Yen and R. Langari, *Fuzzy Logic: Intelligence, Control and Information*, 1st ed.: Upper Saddle River, NJ: Prentice-Hall, 1999.

[98] S. Nazir, S. Shahzad, S. A. Khan, N. B. Ilyas, and S. Anwar, "A novel rules based approach for estimating software birthmark," *Scientific World Journal*, vol. 2015, pp. 1-8, 2015.

[99] S. Nazir, S. Anwar, S. A. Khan, S. Shahzad, M. Ali, R. Amin, M. Nawaz, P. Lazaridis, and J. Cosmas, "Software Component Selection Based on Quality Criteria Using the Analytic Network Process," *Abstract and Applied Analysis*, vol. 2014, pp. 1-12, 2014.

[100] F. J. Cabrerizo, S. Alonso, and E. H.-. Viedma, "A Consensus Model for Group Decision Making Problems with Unbalanced Fuzzy Linguistic Information," *International Journal of Information Technology & Decision Making*, vol. 8, pp. 109-131, 2009.

[101] S. Nazir, S. Shahzad, Q. U. A. Nizamani, R. Amin, M. A. Shah, and A. Keerio, "Identifying Software Features as Birthmark," *Sindh University Research Journal (Science Series)*, vol. 47, pp. 535-540, 2015.

[102] D. G. Zill and M. R. Cullen, *Differential Equations with boundary Value Problem*, 7 ed.: Brooks/Cole Cengage Learning, 2009.

[103] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9 ed. USA: Brooks/Cole, Cengage Learning, 2011.