Chapter 1

Introduction and Statement of the Problem

This chapter provides a short introduction to the thesis. We provide a general motivation to this thesis in section 1.1. In section 1.2, we provide the problem statement. In section 1.3, we list the research objectives and section 1.4 explains the contributions of the thesis in brief. Finally, in section 1.5 we provide an outline of remaining chapters.

1.1 Motivation

Copying existing code fragments and pasting them with or without modifications into other sections of code is very frequent process in software development. The copied code is called a software clone and the process is called software cloning. Software developers tend to copy the existing functionality from the source code and paste it somewhere else either intentionally or unintentionally. The increasing use of open source software and its variants increased code reuse, too. Existing code can be modified to cater to new requirements thereby facilitating and advancing open source development. The importance of clone detection can be gazed from the fact that large percentages of different software artifacts contain duplication as shown in Table 1.1. The table shows different clone detection techniques and subject systems and it reflects the percentage by which the code can be shrunk in software.

Table 1.1 Percentage of cloned code across different systems

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Percentage of cloned code</th>
<th>Subject System</th>
<th>Size (LOC)</th>
<th>Clone Detection Technique</th>
<th>Citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>13%-20%</td>
<td>SS Subsystem</td>
<td>1.1 M</td>
<td>Token based</td>
<td>[12]</td>
</tr>
<tr>
<td>2.</td>
<td>5%-20%</td>
<td>Telecommunication Monitoring Systems</td>
<td>1M</td>
<td>Metrics based</td>
<td>[174]</td>
</tr>
<tr>
<td>3.</td>
<td>12.7%</td>
<td>Process Control System</td>
<td>400K</td>
<td>Tree based</td>
<td>[23]</td>
</tr>
<tr>
<td>4.</td>
<td>50%</td>
<td>gcc and other application projects</td>
<td>6.5K – 460 K</td>
<td>Text based</td>
<td>[52]</td>
</tr>
<tr>
<td>Sr. No.</td>
<td>Percentage of cloned code</td>
<td>Subject System</td>
<td>Size (LOC)</td>
<td>Clone Detection Technique</td>
<td>Citations</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------</td>
<td>----------------</td>
<td>------------</td>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>5.</td>
<td>8.3% - 14.8%</td>
<td>E-Business Website and Resource Management System</td>
<td>15-35</td>
<td>Model based</td>
<td>[158]</td>
</tr>
<tr>
<td>6.</td>
<td>20% - 50%</td>
<td>Java Source Code, C# Source Code</td>
<td>425K, 16K</td>
<td>Tree based</td>
<td>[54]</td>
</tr>
<tr>
<td>7.</td>
<td>14.9%</td>
<td>SRS</td>
<td>2500 pages</td>
<td>Text based</td>
<td>[49]</td>
</tr>
<tr>
<td>8.</td>
<td>12.1% - 32.1%</td>
<td>Open source Python Systems</td>
<td>9KLOC – 272KLOC</td>
<td>Hybrid</td>
<td>[197]</td>
</tr>
</tbody>
</table>

Fowler et al. [58] mentions duplication of code as one of the bad practices in software development increasing maintenance cost. A bug detected in one section of code therefore requires correction in all the replicated fragments of code. Thus, it is important to find all related fragments throughout the source code [168]. Considering the high maintenance cost, software clone detection has emerged as an active research area.

Different programming paradigms and languages have led to number of clone variants and detection techniques.

Different domains like automotive domain, web based applications and other complex systems tend to use models in the development [45]. Model driven software development has been accepted as an industry best practice. Models developed using Unified Modeling Language (UML) improve the quality of product delivery and bring in advantages of automatic code generation, early verification and validation, etc. [143]. As UML models provide an abstract view of the system thus detecting clones in models is equally important because similar challenges as found in source code exist in the case of UML models [216], too. In this thesis, we have proposed a clone detection technique for UML class models which is discussed in chapter 4.

It is true that similarities exist at higher level of abstraction in software which will be referred as concept clones. Concept clones are defined as similar program structures within one software system or across versions reflecting similar domain concepts. Since
the graphical UML [236] is increasingly replacing conventional programming languages for developing and modeling software systems, the presence of design level similarities in form of concept clones cannot be precluded in UML models [105]. Marcus and Maletic [172] identified a scenario in which the software developer writes fresh solution to a problem knowingly or unknowingly for which the solution already exists. This scenario leads to different solutions for the same problem. Identification of high level concept clones helps the software developer in understanding and comprehending the system at abstract level [18]. The presence of structural similarities during the evolution of software system helps in improving the understanding of the system. Chapter 5 of the thesis presents a technique to detect these concept clones.

The detection of software clones will help to raise the awareness of clones existing within systems and aim to prevent the creation of new clones. Other applications of software clone detection include finding cross cutting code [30], code evolution [8], software refactoring [14,76,129,152], plagiarism and copyright infringement detection [28], software product lines [204], clones in web sites [10,147], system quality evaluation [175], locating bugs and reducing software defects [94] and to gain insight in program design and understanding.

1.2 Problem Statement

Code duplication, or cloning, is the use of existing software artifacts in the construction of new code. It is essentially a form of reuse. The cloning leads to spread of an error potentially contained within the duplicated code. Thus, it is important to find all related fragments throughout the source code. Efficient software clone detection depends upon finding fragments of source code with high similarity. In this section, few of the challenges are discussed to formulate the research problem.

1.2.1 General Challenges

A software clone detector needs to compare every fragment with every other fragment which increases the complexity of match detection. Thus, it is an apparent challenge for clone detection. Extensive research has been carried out in the field of source code clone detection. In text based techniques, the source program is considered as sequence of strings or lines and comparison is carried out based upon that only. These techniques yield type-1 clones i.e. exact matches. Token based clone detection techniques yield type-
2 clones i.e. renamed clones. These techniques tokenize the source code and the sequence of tokens is read to trace duplicate subsequences. Similarly type-3 clones are detected by converting the source code to abstract syntax tree (AST). Tree matching or tree similarity techniques are applied to the AST representation of source code. All these techniques of clone detection rely on keyword matching only.

1.2.2 Semantic Clone Detection– A Challenge

Two program fragments with different syntax may be semantically same. These types of clones are known as semantic clones or type-4 clones. It is challenging to detect semantic clones. In 1990, a key article [81] by Horwitz was published to detect textual and semantic similarities.

Program Dependence Graph (PDG) is directed attributed graph representing the statement, control flow and captures semantic information from source code. It is an abstraction of source program. Krinke [137], Komondoor and Horwitz [130] and Gabel et al. [60] presented techniques to detect semantic clones using PDG as source representation. Since PDG is in the form of graph, so graph mining techniques like sub graph isomorphism are applied and detection of isomorphic subgraphs is NP-complete. Choi et al. [36] used a set of API calls to detect similar programs but this technique is vulnerable to deobfuscation attacks, so extending birthmarks with more information and making technique robust against attacks is an upcoming challenge. In Philip Schugerl [205]’s technique, AST is normalized to description logic but the technique currently does not detect small clones and clones across methods. Jiang and Su [97] proposed a technique to detect functionally equivalent code fragments of arbitrary size depending on the input–output behaviour of a piece of code. A general method for different programming languages should be developed to detect functionally equivalent but syntactically different code fragments. Exploring future research on functionally equivalent code refactoring and reuse is another upcoming hot area.

Mostly, various representations adopted for clone detection are closely tied to the source code. These representations encode different smaller changes like reordering, intertwining of clones which usually take place during programming. Semantic clones are particularly hard to detect without a great deal of background knowledge about program construction and software design. As we move from Type-1 Clones to Type- 4 Clones, the
sophistication and complexity underlying corresponding detection increases. PDG based approaches of detecting semantic clones suffer from slow generation of clone pairs and imprecise definition of semantic clones. Developing the framework to help in fast generation of PDG from source code is a definite challenge. Moreover, PDG does not consider statement ordering, thus non-contiguous clones may turn out to be false positives during manual verification. The technique should be scalable seeing the bulk of code in latest software systems which is computationally challenging.

1.2.3 Detection of Clones in Object Oriented Systems

Number of researchers recognized the need to detect semantic equivalence in source code. Due to the assorted nature of the problem, difficulty and cost associated forced them to go for alternate methods which are rather more close to implementation. Moreover, there is a great need to understand the commonalities in domain concepts which most of the time correspond directly to implementations.

Due to the use of model driven development as the standard industry practice, we are motivated to detect clones in object oriented systems by using object oriented i.e. UML models. There is higher number of concept or domain clones than other types of clones [172]. Concepts are usually represented in model driven engineering using general purpose modeling languages like UML. Detection of concept clones can help in comprehending the design of the system for better maintenance [225]. During evolution, high level similarities emerge. The presence of structural similarities as the software system evolves helps in improving the understanding of the system. Detection of high level similarities throughout the history of the subject system helps in improving the different attributes of quality of the software like maintainability, reusability, extensibility [110].

Clone detection in models [45] [187] [158] is touched vastly by related fields like algorithms, symbolic logic, data mining, graph mining, artificial intelligence, programming languages, machine learning, etc. Moreover large systems introduce intricacies in the form of unexpected overlaps of parts and unexpected behavior of previously well behaved systems [236] as many designs can be easily shared between design teams [118]. Tim Weilkiens [236] confirms that the design methodology of any system or process has shared parts as crucial component. As the UML model of a system is generated before implementation, most of the reasons leading to clones in code based
development are also applicable to model based development. Duplications in models have negative effects on maintainability and reusability [216] and thus the study states the importance of detecting clones in models to improve quality. The detection of model clones is an active area of research [215]. There exists only one approach [216] to detect clones in UML models. No study has been carried out to explicitly examine the effects of cloning in model based development.

1.3 Research Objectives

It is common consensus and utmost important to find software clones. We understand that the presence of software clones hamper maintenance and may lead to bug propagation in source code. The detection of clones will help to raise the awareness of clones existing within the system and aim to prevent the creation of new clones. With the popularity of model driven development, duplicate parts in models i.e. model clones pose similar challenges as in source code. Thus the complete problem is divided into following major objectives:

1. To explore various software clone detection techniques
2. To identify various parameters affecting software semantic clone detection
3. To design and develop an efficient software clone detection technique for object-oriented systems
4. To verify and validate the clone detection technique

1.4 Contributions of the Thesis

We have conducted an extensive systematic literature review of software clones in general and software clone detection in particular. We used the standard systematic literature review method based on a comprehensive set of 213 articles from a total of 2039 articles published in 11 leading journals and 37 premier conferences and workshops. Existing literature about software clones is classified broadly into different categories. The importance of semantic clone detection and model based clone detection led to different classifications. Empirical evaluation of clone detection tools/techniques is presented. Clone management, its benefits and cross cutting nature is reported. Number of studies pertaining to nine different types of clones is reported. 13 intermediate representations and 24 match detection techniques are reported. We call for an increased
awareness of the potential benefits of software clone management, and identify the need to develop semantic and model clone detection techniques. Recommendations are given for future research.

Now-a-days, model driven development has become a standard industry practice. Duplicate parts in models i.e. model clones create similar problems as in source code. We have developed a technique to detect clones in UML class models. The core of our technique is the construction of a labeled, ranked tree corresponding to the UML class model where attributes with their data types and methods with their signatures are represented as subtrees. By grouping and clustering of repeating subtrees, the tool is able to detect duplications in a UML class model at different levels of granularity i.e. complete class diagram, attributes with their data types and methods with their signatures across the model and cluster of such attributes/methods. We propose a new classification of model clones with the objective of detecting exact and meaningful clones. Empirical evaluation of the tool using open source reverse engineered and forward designed models show some interesting and relevant clones that provide useful insights into software modeling practice.

We have developed another technique to compute similarity across object oriented programs at different levels of granularity. The tool is able to detect concept level similarities by applying latent semantic indexing and principal component analysis. Previous research has shown that detection of high level similarities can help in comprehending the design of the system for better maintenance. We have extended our tool to detect similarities for UML diagrams by measuring the distance between two class models. In addition, we mined important change patterns at method level using multi-version program analysis by applying the proposed technique throughout the evolutionary history of the software. We have validated the similarity score by applying the tool at function level granularity in the source code. We assess the usefulness and scalability of the proposed technique by empirical evaluation on source code of open source subject systems and multi-version program analysis on 8 releases of dnsjava.
1.5 Organization of the Thesis

The remaining chapters of this thesis are organized as follows:

- The second chapter provides systematic literature review of existing work related to software clone detection. This chapter starts with background information which includes types of clones, reasons for occurrence of clones, etc. We have done rigorous study of the existing techniques of software clone detection with a complete list of available clone detection tools. This chapter also identifies the comparison and evaluation studies. Key sub areas related to software clone detection with particular emphasis on clone management is also mentioned in this chapter.

- Chapter 3 summarizes all the semantic clone detection techniques and model based clone detection techniques. Various intermediate source representations and match detection techniques are listed in comparing these techniques. We have also highlighted the process of identifying the parameters of software clone detection. The chapter concludes with general motivation for model clone detection.

- In chapter 4, we devised the clone detection technique for UML models. Various reverse engineered and forward designed models are used for empirical evaluation to verify and validate the technique.

- Chapter 5 presents another approach in which we detected high level similarities in the form of concept clone detection in source code and concept model clone detection in UML models. The results of clone detection at different levels of granularity are also mentioned.

- Finally, chapter 6 concludes the thesis along with directions for future research.