Chapter 1

Introduction

- Determining the detailed structural similarities and differences between entities of the source code of a software system, or between the source code of different software systems, is a potentially complex problem
- Being able to do so has various actual or potential applications:
  - code clone detection :
  - semi-automating source code reuse [Cottrell et al. ,2008]
  - recommending replacements for an API between various versions of a software library [Cossette et al., 2014]
  - collating API usage patterns
  - automating the merge operation of various branches in a version control system
- As a specific application, our focus is on the study of where logging is used in the source code
- Logging is a conventional programming practice to record an application's state and/or actions during the program's execution
- Logging is a pervasive practice during software development [Yuan et al., 2012]
- The importance of logging has been identified by its various applications in:
  - problem diagnosis
  - system behavioural understanding
  - quick debugging
  - performance diagnosis

  - easy software maintenance
  - troubleshooting

- Researchers have often considered logging as a trivial task (AOP/AOSD literature)
- However, some evidence suggest that it is not a straightforward task to perform high quality logging in practice, such as:
  - availability of several complex frameworks to help developers to log
  - the significant amount of effort developers spend to modify logging calls as after-thoughts [Yuan et al., 2012]
- Developers have to make decisions about where and what to log
- Logging should be done in an appropriate manner to be effective
  - Excessive logging:
    - can generate a lot of redundant information, masking significant ones for system log analysis
    - requires extra time and effort to write, debug, and maintain the logging code
    - can cause system resource overhead
  - Bad usage of logging can affect the application's performance
  - Insufficient logging may result in losing some necessary run-time information for software analysis
- So far, little study has been conducted on characterizing the usage of logging in real-world systems
- PROBLEM: In this research, we would like to understand where developers log in practice, in a detailed way

- The location of log statements has a great impact on the quality of logging since it helps developers to trace the code execution path to identify the root causes of errors in log system analysis

- SOLUTION: Develop an automated approach to detect the detailed structural similarities and differences in the usage of logging within a system and between systems

- **[NZ: I am a little confused about how to find commonalities and differences of logging usage between systems? do you mean that I should compare the results taken from clustering of all Java classes in one system to another one?] [RW: The process of locating commonalities and differences can be applied within a single version of one system, across multiple versions of one system, across single versions of multiple systems, or across multiple versions of multiple systems. It would be useful to understand the differences and similarities between different systems. Is there some reason that this would be harder to achieve?] [NZ: I should think about it more. I can answer to this question after per-system analysis with more confidence]**

  [RW: I AM WORRIED ABOUT WHAT YOU HAVE BEEN SAYING HERE. THE FACT THAT TWO LOGGING STATEMENTS COME FROM TWO SYSTEMS INSTEAD OF FROM ONE SYSTEM SHOULD MAKE ZERO DIFFERENCE FROM THE ALGORITHMIC PERSPECTIVE. THAT YOU THINK OTHERWISE MAKES ME WONDER IF YOU HAVE MISUNDERSTOOD SOMETHING.]

1.1 Broad thesis overview

- We aim to provide a concise description of where logging calls are used in the source code through creating generalizations that represents the detailed structural similarities and differences between LJCs

- Our approach:

  - applies a hierarchical clustering algorithm to classify LJCs into groups using a measure of similarity
  - uses an antiunification algorithm to construct a structural generalization representing the similarities and differences of all LJCs in each group

- Our antiunification approach:
  - uses the Jigsaw framework to determine all potential correspondences between a pair of LJCs
  - applies some constraints to avoid antiunifying logged Java classes with non-logged Java classes
  - determines correspondences between structures containing logging calls by greedily applying a similarity measure to find the most similar substructures
  - constructs an antiunifier
  - develops a measure of structural similarity between LJCs
- Our approach has been implemented as an Eclipse plug-in, which is evaluated by conducting an empirical study on 10 sample logged Java classes
- Our tool has been applied on the source code of three open-source software systems that make use of logging
- Our tool extracts all logged Java classes from these systems to construct the structural generalizations
- Our evaluation shows ...

1.2 Overview of related work

- Yuan et al. [2012] provides a quantitative characteristic study of log messages on four open-source software system, however, it does not study the location of logging calls in the source code

- So far, antiunification has been used for various applications:
  - to construct a generalized correspondence view of two source code fragments [Cottrell et al., 2007]
  - to help developers to perform small-scale reuse tasks semi-automatically[Cottrell et al., 2008]
  - Software clone detection [Bulychev and Minea, 2008]
- This study makes the first attempt to characterize where logging calls occur in the source code through finding the detailed structural similarities and differences using HOAUMT

1.3 Thesis Statement

The thesis of this work is to determine the detailed structural similarities and differences between entities of the source code that make use of logging to provide a concise description of where logging do occur in real systems

1.4 Thesis Organization

- Chapter **??** : motivates the problem of understanding where to use logging calls in the source code through an example
- Chapter **??**: provides background information on:
  - abstract syntax trees (ASTs), which are the basic structure we will use for describing software source code
  - how ASTs are realized in the Eclipse integrated development environment, the industrial tool we will build atop
  - antiunification and its limitations to solve our problem context

  - higher-order antiunification modulo theories (HOAUMT) that can be applied on an extended form of the AST structure to address our problem
  - the Jigsaw framework, an existing tool for a subset of HOAUMT, which we extend to address our problem **[RW: Why “a subset”?]**

- Chapter **??**: describes our proposed approach and its implementation as an Eclipse plug-in
- Chapter **??**: presents an empirical study conducted to evaluate our approach and its application to characterize logging usage
- Chapter **??**: discusses the results and findings of my work, threats to its validity, and the remaining issues.
- Chapter **??**: describes related work to our research problem and how it does not adequately address the problem
- Chapter **??**: concludes the dissertation and presents the contributions of this study and future work