e-Informatica Software Engineering Journal, [redacted]

ECLogger: Cross-Project Catch-Block Logging Prediction using Ensemble of Classifiers

Sangeeta Lal*, Neetu Sardana*, Ashish Sureka**
*Jaypee Institute of Information Technology, Noida, Uttar-Pradesh, India
**ABB Corporate Research, Bangalore, India
sangeeta@jiit.ac.in, neetu.sardana@jiit.ac.in, ashish.sureka@in.abb.com

Abstract

Background: Software developers insert log statements in the source code to record program execution information. However, optimizing the number of log statements in the source code is challenging. Machine learning based within-project logging prediction tools, proposed in previous studies, may not be suitable for new or small software projects. As these software projects do not have sufficient prior training data to construct the prediction model. For such software projects, we can use cross-project logging prediction. **Aim:** The aim of the study presented is paper is to investigate cross-project logging prediction methods and techniques. **Method:** We propose *ECLogger*, which is a novel, ensemble-based, cross-project, catch-block logging prediction model. We use 9 base classifiers and combine them using 3 ensemble techniques to improve the cross-project logging prediction result. We evaluate the performance of ECLogger on three open-source Java projects: Tomcat, CloudStack and Hadoop. **Results:** ECLogger$_{Bagging}$, ECLogger$_{AverageVote}$, and ECLogger$_{MajorityVote}$ show a considerable improvement in the average LF in 3, 5, and 4 source→target project pairs, respectively, compared to the baseline classifiers. ECLogger$_{AverageVote}$ performs the best and shows improvements of 3.12% (average LF) and 6.08% (average ACC) compared to the baseline classifier. **Conclusion:** The classifier based on ensemble techniques such as bagging, average vote and majority vote outperforms the baseline classifier. Overall, the ECLogger$_{AverageVote}$ model performs the best.The results show that the CloudStack project is more generalizable than the other projects.

1. Introduction

Logging is an important software development practice that is typically performed by inserting log statements in the source code. Logging helps in tracing the program execution. In the case of failure, software developers can use this tracing information to debug the source code. Logging is important because this is often the only information available to the developers for debugging because of problems in recreating the same execution environment or because of unavailability of the input used (security/privacy concerns of the user). Logging statements have many applications, such as debugging [81] workload modelling [66], performance problem diagnosis [55], anomaly detection [25], test analysis [33, 34], and remote issue resolution [2].

Source code logging is important, but it has a trade-off between cost and benefit [26, 43, 44, 89]. Excessive logging in the source code can cause performance and cost overhead. It can also decrease the benefits of logging by generating too many trivial logs, which can potentially make debugging