Chapter 3
Logging Analysis and Prediction in Open Source Java Project

Sangeeta Lal
Jaypee Institute of Information Technology, India

Neetu Sardana
Jaypee Institute of Information Technology, India

Ashish Sureka
Ashoka University, India

ABSTRACT

Log statements present in source code provide important information to the software developers because they are useful in various software development activities such as debugging, anomaly detection, and remote issue resolution. Most of the previous studies on logging analysis and prediction provide insights and results after analyzing only a few code constructs. In this chapter, the authors perform an in-depth, focused, and large-scale analysis of logging code constructs at two levels: the file level and catch-blocks level. They answer several research questions related to statistical and content analysis. Statistical and content analysis reveals the presence of differentiating properties among logged and nonlogged code constructs. Based on these findings, the authors propose a machine-learning-based model for catch-blocks logging prediction. The machine-learning-based model is found to be effective in catch-blocks logging prediction.

INTRODUCTION

Logging is an important software development practice that is used to record important program execution points in the source code. The recorded log generated from program execution provides important information to the software developers at the time of debugging. Fu et al. (2014) conducted a survey of Microsoft developers, asking them their opinion on source code logging. Results of the survey showed that 96 percent of the developers consider logging statements the primary source of information for problem diagnosis. In many scenarios, logging is the only information available to the software developers

DOI: 10.4018/978-1-5225-5314-4.ch003

for debugging because the same execution environment is unavailable (which makes bug regeneration difficult) or the same user input is unavailable (because of security and privacy concerns) (Yuan et al., 2012). Yuan et al. (2012) showed in their characterization study that the bug reports consisting of logging statements get fixed 2.2 times faster compared to the bug reports not consisting of any logging statements. Logging statements are not only useful in debugging, but they are also useful in many other applications, such as anomaly detection (Fu et al., 2009), performance problem diagnosis (Nagaraj et al., 2012), and workload modeling (Sharma et al., 2011).

Logging statements are important, but they have an inherent cost and benefit tradeoff (Fu et al., 2014). A large number of logging statements can affect system performance because logging is an I/O-intensive activity. An experiment by Ding et al. (2015) and Sigelman et al. (2010) reveal that in the case of search engines, logging can increase average execution time of requests by of 16.3%. Similar to excess logging, less logging is also problematic. An insufficient number of logging statements can miss important debugging information and can lessen the benefits of logging. Hence, developers need to avoid both excessive and insufficient logging. However, previous research and studies show that developers often face difficulty in optimal logging, that is, identifying which code construct to log in the source code (Fu et al., 2014; Zhu et al., 2015). It happens because of lack of training and the domain experience required for optimal logging. For example, Shang et al. (2015) reported an incident of a user from a Hadoop project complaining about less logging of catch-blocks. Recently the software engineering research community has conducted studies to understand the logging practices of software developers in order to build tools and techniques to help with automated logging. The current studies provide limited characterization study or conduct analysis on fewer code constructs. There are gaps in previous studies, as they do not analyze all the code constructs in detail, which this study aims to fill.

The work presented in this chapter is the first large-scale, in-depth, and focused study of logged and nonlogged code constructs at multiple levels. High-level (source code files) and low-level (catch-blocks) analysis were conducted to identify relationships between code constructs and logging characteristics. Based on the finding of this multilevel analysis authors proposed a machine leanirng based model for log statement prediction for catch-blocks. . A case study was performed on three large, open-source Java projects: Apache Tomcat (Apache Tomcat, n.d.), CloudStack (Apache CloudStack, n.d.), and Hadoop (Page, n.d.). Empirical analysis reveals several interesting insights about logged and nonlogged code constructs at both the levels. The machine learning based model give encouraging results for catch-blocks logging prediction on Java projects.

RELATED WORK

This section presents the closely related work and the novel research contributions of the study presented in this chapter in context to existing work. The authors categorize the related work in three dimensions: 1) improving source code logging, 2) uses of logging statements in other applications, and 3) applications of LDA in topic identification.

Improving Source Code Logging

Yuan et al. (2012) analyze source code and propose *ErrorLog* tool that logs all the generic exception patterns. However, logging all the generic exception can cause excess of log statements in the source