2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

Improving the software logging practices in DevOps

Boyuan Chen
York University, Toronto, Canada
chenfsd@eecs.yorku.ca

***Abstract*—DevOps refers to a set of practices dedicated to accelerating modern software engineering process. It breaks the barriers between software development and IT operations and aims to produce and maintain high quality software systems. Software logging is widely used in DevOps. However, there are few guidelines and tool support for composing high quality logging code and current application context of log analysis is very limited with respect to feedback for developers and correlations among other telemetry data. This thesis proposes automated approaches to improving software logging practices in DevOps by leveraging various types of software repositories (e.g., historical, communication, bug, and runtime repositories). We aim to support the software development side by providing guidelines and tools on developing and maintaining high quality logging code. We aim to support the IT operation side by enriching the log analysis context through systematic estimating code coverage via executing logs and in-depth problem diagnosis by correlating logs with other telemetry data (e.g., traces and APM data). Case studies show that our approaches can provide useful software logging suggestions to both developers and operators in open source and commercial systems.**

I. Introduction

DevOps is a software development methodology that intends to automate the process between software development and IT operations. The goal is to reduce the time between committing a change to a system and placing it to production, while ensuring high quality [1]. Compared to traditional software development process, DevOps provides faster feedback between software development and IT operations so that new features and bug fixes can be released faster to the customers. To ensure the quality and the health of the deployed systems, software logging plays a central role.

Software logging in the context of DevOps refers to the practices of developing and maintaining logging code and analyzing the resulting execution logs. Logging code refers to the code snippets that developers inserted into source code (e.g., `LOG.info("User " + userName + " logged in")`) to monitor the behavior of systems during runtime. There are typically four types of components in a snippet of logging code: a logging object, a verbosity level, static texts, and dynamic contents. In the above example, the logging object is `Logger`, the verbosity level is `info`, the static texts are `User` and `logged in`, and the dynamic content is `userName`. Execution logs (a.k.a., logs), which are generated by logging code during runtime, are readily available in large-scale software systems for many purposes like system monitoring [2], problem debugging [3], workload characterization [4], and business decision making [5]. Stale or incorrect logging code may cause confusion [6] or even more serious issues like system crash [7]. In particular, there are four major challenges associated with the software logging practices in DevOps:

- C_1**: No existing guidelines on producing high quality logging code.** Recent empirical studies show that there are no existing logging guidelines for commercial [8] and open source systems [9], [10]. Developers write logging code solely based on domain expertise and revise them in an ad-hoc fashion [9], [10]. Unlike feature code, which can be examined through testing, it is very challenging to verify the correctness of logging code.
- C_2**: Difficulty in maintaining and evolving logging code.** As logging code tangles with source code, it is very challenging to maintain and update logging code along with feature code for constantly evolving systems. Although there are language extensions (e.g., AspectJ [11]) to support better management of logging code, many industrial and open source systems still choose to intermix logging code with feature code [9], [10].
- C_3**: Limited mechanism for quality feedback.** In the context of DevOps, the software testing process is completely changed compared to traditional software development process, as many testing activities are automated and occur in the field [12]. There is limited mechanism for quality feedback from the IT operation to the software development. This problem becomes even more serious, as in DevOps code base evolves more rapidly with usage scenarios being constantly added or modified.
- C_4**: Heterogeneous and complex telemetry data.** Besides execution logs, large scale distributed systems also adopt other mechanisms to monitor the health of systems. Some examples are distributed tracing [13] and Application Performance Monitoring (APM) tools [14]. Existing problem diagnosis techniques only focus on one type of telemetry data (e.g., logs [15] or traces [16]). Very few works try to enrich the analysis by correlating the information among different types of telemetry data.

Motivated by the importance and challenges, throughout the thesis, we propose systematic approaches to improving the software logging practices to aid software development and IT operations by leveraging various types of software repositories. Our overall process is shown in Figure 1. For each challenge, we list a corresponding anticipated research outcome. We will further describe them in more details in Section IV.

The rest of this paper is organized as follows. Section II describes the current research on software logging. Section III presents our research hypothesis and Section IV explains our

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00080

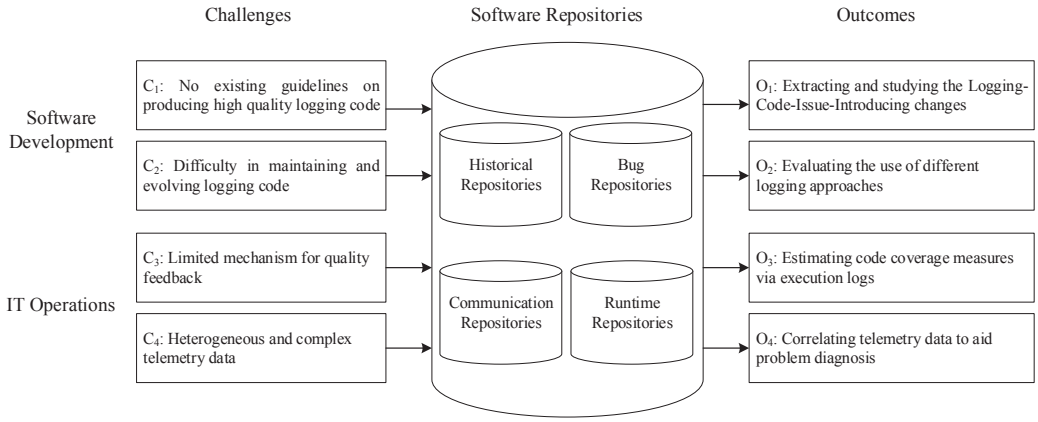

Fig. 1: Overall process

approaches. Section V lists the expected thesis contributions and Section VI concludes this paper.

II. Current Research on Software Logging

In this section, we describe the current research on software logging. We separate them in two aspects: logging in the context of software development and IT operations.

A. Logging for Software Development

The current research works on developing high quality logging code can be categorized into three types: *what-to-log*, *where-to-log*, and *how-to-log*.

The problem of *what-to-log* is about providing sufficient information in logging code. Yuan et al. [17] proposed an approach based on program analysis to adding variables to existing logging code for assisting error diagnosis. He et al. [18] systematically characterized natural language descriptions used in logging code and shed lights on automated logging description generation.

The problem of *where-to-log* is about deciding the appropriate logging points. Yuan et al. [15] proposed an approach based on program analysis to inferring additional logging points. Zhao et al. [19] introduced Log20, which can automate logging code placement under certain overhead threshold. Fu et al. [8], [20] proposed an approach based on data mining to identifying the important factors impacting locations of logging points. Cinque et al. [21] proposed a logging method based on a set of rules, which can be used to detect software failures by logs.

The problem of *how-to-log* is about developing and maintaining high quality logging code. There are few works in this area except [22], [23], which focus on a few manually analyzed samples. In this thesis, we aim to provide benchmarks for detecting issues in logging code and better tool support for managing logging code during software evolution process.

B. Logging for IT Operations

Logging is used pervasively in IT operations including monitoring [2], debugging [3], workload characterization [4], and business decision making [5]. Xu et al. [2] proposed an anomaly detection technique to flag problematic behavior by mining the generated logs. Oliner et al. [3] demonstrated that logs can be used to debug system performance. Hassan et al. [4] leveraged data compression to characterize workload from execution logs. Barik et al. [5] highlighted the importance of logging in data-driven decision making at Microsoft.

In the context of DevOps, IT operators are tasked with new responsibilities. In particular, the software testing process is completely changed in DevOps with many testing activities occurring in the field [12]. Unfortunately, there are very few works about providing systematic quality feedback to the software development side. Furthermore, many of the existing log analysis works only focus on the execution logs alone. But logs are just one type of the telemetry data existed in the field. Other types of telemetry data like traces or APM data can provide additional insight for the system behavior during runtime. However, little research is done to enrich the log analysis by correlating their information together.

III. Research Hypothesis

Software repositories (e.g., code repositories, communication repositories, runtime repositories, and bug repositories) which are readily available and contain rich information about software development and system behavior during runtime, can be leveraged to systematically improve the software logging practices in the context of DevOps.

We mainly rely on four types of repositories to tackle challenges in software logging practices. The *historical repositories* refer to the source code version control systems like GitHub and SVN. The *communication repositories* refer to the online communication data from StackOverflow and developer mailing list. The *runtime repositories* refer to the telemetry data generated in various scenarios. The *bug repositories* refer to the issue tracking systems such as JIRA and BugZilla. In the thesis, we attempt to improve logging from two dimensions: software development and IT operation. For the aforementioned four challenges (C_1 - C_4), we will propose corresponding research outcomes (O_1 - O_4), which address

these challenges. O_1 and O_2 address the first two challenges which are on the development side. O_3 and O_4 address the last two challenges which are on the IT operation side.

- (O_1): We will mine the historical and bug repositories to extract a benchmark dataset which contains real-wold issues in logging code so that interested researchers could develop and evaluate their techniques of automated detection of logging code issues.
- (O_2): We will mine the communication and bug repositories to compare various types of logging approaches (e.g., ad-hoc, centralized, and aspect-oriented logging).
- (O_3): We will propose automated techniques to estimate code coverage measures(e.g., statement coverage, branch coverage) by correlating source code with the logs stored in the runtime repositories.
- (O_4): We will combine various types of telemetry data (e.g., logs and tracing data) stored in the runtime repositories to perform in-depth problem diagnosis.

IV. Our Approach

In this section, we describe our approaches to tackling the aforementioned four challenges.

O_1: Extracting and Studying the Logging-Code-Issue-Introducing Changes

Incorrect or outdated logging code may cause confusion [6] or even more serious problems like crashes [7]. Our previous work [22] is the first work tackling the problem of *how-to-log* by deriving anti-patterns (a.k.a., common mistakes) in logging code. However, it only focuses on a few hundred manually examined code snippets.

We will develop a general approach to extracting the issues in logging code from different projects. First, we will systematically analyze the commit logs from the historical repositories and the descriptions of the bug reports to derive a list of code commits, which are related to the fixes to various issues in the logging code. The four components in logging code (the logging object, the verbosity level, the static texts, and the dynamic contents) can be changed separately in different revisions. The traditional approach of automatically locating bug introducing changes from their fixes (a.k.a., the SZZ algorithm [24]) would not work, as it treats each line of change as one single entity. To cope with this problem, we will develop an adapted version of the SZZ algorithm, called LCC-SZZ (Logging Code Change-based SZZ), to automatically locate the Logging-Code-Issue-Introducing changes from their fixes. Once we have derived such a benchmark dataset, we will also perform an exploratory study to evaluate the effectiveness of various existing logging code issue detection tools.

Expected timeline: This work has just been accepted by the journal of Empirical Software Engineering in January 2019.

O_2: Evaluating the Use of Different Logging Approaches

Although there are three general approaches to composing and managing logging code: manual ad-hoc logging, centralized logging, and aspect-oriented logging, existing studies [9], [10] show that most of the systems, including many well-maintained projects, still chose the manual ad-hoc approaches to develop, maintain, and evolve their logging code. It is not clear why developers decide against systematic (i.e., centralized logging) or automated (i.e., AOP) logging approaches.

This research will be carried out in two aspects. First, we will survey the developer mailing lists, bug reports, commit logs, and online posts (e.g., Stackoverflow) in terms of pros and cons of various logging approaches. Then, we will study the logging practices by analyzing the logging code data from well-maintained projects in the historical repositories (e.g., Apache Software Foundation and GitHub). We intend to characterize and transform the existing logging code written in the ad-hoc logging approach to the centralized or automated logging approaches. This transformation experiment will be conducted in two contexts: (1) for all the logging code in a release snapshot, and (2) for logging code snippets across time during software maintenance. During this process, we will record the efforts that we spend as well as the difficulties and issues that we encounter. We anticipate this work will be useful to programming language designers or developers who are responsible for maintaining and evolving logging code.

Expected timeline: This work is currently at the exploratory stage. We plan to finish this work in 2020.

O_3: Estimating Code Coverage Measures via Execution Logs

It is very challenging to ensure the validity and representativeness of test cases in DevOps, as traditional code coverage tools suffer from three major issues: engineering challenges, performance overhead, and incomplete results.

We will propose an automated technique of estimating code coverage measures via execution logs. We will first analyze the system's source code and derive a list of possible code paths and their corresponding log sequences, which will be matched with the execution logs. Based on the matched results, we label the code regions as `Must` (definitely covered), `May` (maybe covered, maybe not), and `Must-not` (definitely not covered) and use these labels to infer three types of code coverage criteria: method coverage, statement coverage, and branch coverage. We will evaluate our techniques on commercial and open source software projects from two dimensions: accuracy (a.k.a., comparing against existing code coverage tools) and usefulness (e.g., comparing the code coverage results between the field and the existing test cases).

Expected timeline: This work has been published in ASE 2018 [25]. Case studies on one open source system (HBase) and five commercial systems from Baidu show that: (1) the results of our research prototype, LogCoCo (Log-based Code Coverage), is highly accurate under various testing activities (unit testing, integration testing, and benchmarking), and (2) the results of LogCoCo can be used to evaluate and improve the existing test suites. Our collaborators at Baidu are currently considering adopting LogCoCo on a daily basis.

O_4: Correlating Telemetry Data to Aid Problem Diagnosis

Existing problem diagnosis techniques only focus on one type of the telemetry data (e.g., logs [15] or traces [16]).

Very few works try to enrich the analysis by correlating the information among different types of telemetry data.

We will conduct an empirical study on the use of the various monitoring data in the open source and commercial systems. We will scan through the source code to identify their use cases. Then, we will examine the existing bug reports and characterize the types of reported real-world problems, the required monitoring artifacts (e.g., only analyzing the logs, or correlating logs and APM data) in order to perform the diagnosis, and their analysis approaches. We seek to provide automated tool support to correlate and analyze various telemetry data when recurrent or similar problems surface.

Expected timeline: This work is currently at the exploratory stage. We plan to finish this work by May 2021.

V. Expected Thesis Contributions

The following are our expected thesis contributions:

1) **Guidelines and tool support for developing and maintaining high quality logging code**: we will provide a benchmark dataset to aid software engineering researchers to derive the best logging practices and develop automated techniques to flag issues in logging code. We will also identify the gaps between the various needs in software logging and the missing functionalities in the existing log management tools.
2) **Expanding and enriching log analysis techniques for wider application context**: compared to traditional code coverage tools, which are impractical to be used in the DevOps context due to the issues of deployment difficulty, performance overhead and result incompleteness; our approach to estimating code coverage measures via execution logs [25] expands the application context of log analysis. As many field problems are still diagnosed manually and dealt in a case-by-case manner, our proposed technique to automatically correlate various telemetry data can significantly enrich the log analysis context and reduce the manual analysis effort.

VI. Conclusions

Although software logging plays a key role in DevOps, there are various challenges associated with it. This thesis aims to overcome the challenges and improve existing software logging practices through systematic analysis of software repositories. We will derive guidelines for developing high quality logging code and empirically evaluate various approaches associated with logging code management. We will analyze logs to provide quality feedback to the developers and perform deeper problem analysis by combining logs with other telemetry data. The resulting findings and techniques will be beneficial for both software developers and IT operators.

References

[1] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, 1st ed. Addison-Wesley Professional, 2015.

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, 2009.

[3] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, Feb 2012.

[4] A. E. Hassan, D. J. Martin, P. Flora, P. Mansfield, and D. Dietz, "An industrial case study of customizing operational profiles using log compression," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, 2008.

[5] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The Bones of the System: A Case Study of Logging and Telemetry at Microsoft," in *Companion Proceedings of the 38th International Conference on Software Engineering*, 2016.

[6] W. Shang, M. Nagappan, A. E. Hassan, and Z. M. Jiang, "Understanding Log Lines Using Development Knowledge," in *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2014.

[7] "HBASE-750: NPE caused by StoreFileScanner.updateReaders," https://issues.apache.org/jira/browse/HBASE-750/, Last accessed: 11/18/2018.

[8] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where Do Developers Log? An Empirical Study on Logging Practices in Industry," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014.

[9] B. Chen and Z. M. Jiang, "Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation," *Empirical Software Engineering*, 2017.

[10] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012.

[11] "AspectJ," https://eclipse.org/aspectj/, Last accessed: 11/18/2018.

[12] B. Adams and S. McIntosh, "Modern Release Engineering in a Nutshell – Why Researchers Should Care," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, 2016.

[13] "Opentracing," https://opentracing.io, Last accessed 2018/10/24.

[14] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang, "Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report," in *Proceedings of the 13th International Conference on Mining Software Repositories (MSR)*, 2016.

[15] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be Conservative: Enhancing Failure Diagnosis with Proactive Logging," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2012.

[16] S. Grant, H. Cech, and I. Beschastnikh, "Inferring and Asserting Distributed System Invariants," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018.

[17] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[18] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018.

[19] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou, "Log20: Fully Automated Optimal Placement of Log Printing Statements Under Specified Overhead Threshold," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017.

[20] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *Proceedings of the 37th International Conference on Software Engineering*, 2015.

[21] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2013.

[22] B. Chen and Z. M. Jiang, "Characterizing and detecting anti-patterns in the logging code," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 71–81.

[23] M. Hassani, W. Shang, E. Shihab, and N. Tsantalis, "Studying and detecting log-related issues," *Empirical Software Engineering*, 2018.

[24] J. Śliwerski, T. Zimmermann, and A. Zeller, "When Do Changes Induce Fixes?" in *Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR)*, 2005.

[25] B. Chen, J. Song, P. Xu, X. Hu, and Z. M. J. Jiang, "An automated approach to estimating code coverage measures via execution logs," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 305–316.