Bridging the Divide between Software Developers and Operators Using Logs

Weiyi Shang
Software Analysis and Intelligence Lab (SAIL), Queen's University, Canada
swy@cs.queensu.ca

***Abstract*—There is a growing gap between the software development and operation worlds. Software developers rarely divulge development knowledge about the software to operators, while operators rarely communicate field knowledge to developers. To improve the quality and reduce the operational cost of large-scale software systems, bridging the gap between these two worlds is essential. This thesis proposes the use of logs as mechanism to bridge the gap between these two worlds. Logs are messages generated from statements inserted by developers in the source code and are often used by operators for monitoring the field operation of a system. However, the rich knowledge in logs has not yet been fully used because of their non-structured nature, their large scale, and the use of the ad hoc log analysis techniques. Through case studies on large commercial and open source systems, we plan to demonstrate the value of logs as a tool to support developers and operators.**

I. Introduction

Software developers are focused on developing feature-rich and bug-free software, while software operators are focused on ensuring a failure-free and scalable operation of the software. In current practice, there is a gap between software developers and operators. Software developers are rarely given access to field knowledge (i.e., information about the real-field deployments), while operators are rarely aware of the development knowledge (e.g., internal details about new features). For instance, developers need field knowledge to understand whether their design and implementation perform well in the field, while operators need development knowledge to help them resolve operational problems. If development teams are aware that a particular piece of code is critical based on field executions, then they are more likely to improve the code and assign it to more senior developers. If operators have more in-depth knowledge about the design or the inner-meaning of error messages, they might be able to resolve problems in a timely fashion without needing to wait for the intervention of developers.

Recent efforts [1] share our concerns about the divide between these two worlds and have proposed the need to bridge these two worlds through better documentation and communication channels. We believe that logs are an ideal medium to bridge the two worlds. Today, many large-scale software systems produce gigabytes or terabytes of logs on an hourly or daily basis, due to various legal and operational requirements [2]. Logs are explicitly introduced due to 1) developers needing to track information that is relevant from a developmental point of view, or 2) operators needing to track information that is relevant from an operational point of view. Over the years, the logs contain a wealth of information based on the needs of both worlds. Yet, these logs are not used in a systematic manner due to:

- The large scale of field logs. This scale impacts the field performance. It complicates and slows down the analysis.
- The non-structured nature of logs compared to source code. The logs are often analyzed using ad hoc unmaintainable scripts.
- The limited awareness of the importance of logs by either world. Developers feel they own the logs so they are often changing them and updating them, while operators are always updating their scripts to cope with such changes.

In this thesis, we propose several techniques that make use of logs to support developers and operators. To overcome the unstructured nature, the ad hoc processes and the scalability challenges of logs, we propose to use a scalable relational algebra, built on top of a web mining framework (i.e., Pig [3]). We will demonstrate our proposed contributions using several open source (e.g., Hadoop) and commercial systems.

The rest of this paper is organized as follows: We present the research hypothesis in Section II. A pilot empirical study is presented in Section III. We propose approaches to validate our research hypothesis in Section IV. In Section V, we discuss the challenges of using our approach and how to address the challenges. In Section VI, we present the state of the art in practice. Section VII concludes our work.

II. Research Hypothesis

Logs are a rich source of information about the development and operations of software systems. Yet today, this source is rarely leveraged in a systematic manner by developers and operators of large systems. The development of systematic and scalable log processing approaches will lead to the improvement of the development and operational quality of software systems.

We identify two sources of knowledge about software: development knowledge and field knowledge. Development knowledge, typically owned by developers, corresponds to the historical information about the development of the software. It resides in software historical data such as historical

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

ICSE 2012, Zurich, Switzerland
Doctoral Symposium

code changes and bug reports. Field knowledge, typically owned by operators, corresponds to the information about how the system operates in the field. Throughout the thesis, we attempt to bridge the gap between software developers and operators by mining large-scale logs:

- For developers, we leverage field knowledge in logs to improve the quality of software. 1) We propose techniques to identify error-prone software components based on log maintenance history. 2) We propose techniques to empirically measure the field-test coverage for large-scale systems.
- For operators, we leverage development knowledge in logs to cope with the operational complexity of software. 1) We propose techniques to automatically document and explain log lines. 2) We propose techniques to reduce logs using development history.

III. An Exploratory Study of the Evolution of Logs

As a pilot study, we performed a study on the execution logs of 10 releases of an open source software system named *Hadoop* and 9 releases of a legacy enterprise application [4]. Our goal of this study is to see: 1) whether logs are static or they evolve with code; 2) what is the rationale for the evolution of logs; and 3) whether the evolution of logs considers the needs of operators. We found that:

- Logs change at a rather high rate across versions, leading to more work for operators who have already built an ecosystem of *Log Processing Apps* around logs.
- 40% to 60% of these changes are not needed and can be avoided. For example, simple rewording of log lines by developers which lead to problems for operators who have written analysis scripts.
- The impact of 15% to 50% of the changes can be controlled through the use of the robust analysis techniques.
- Logs that communicate development knowledge (e.g., low-level implementation details) change more often than logs that communicate operational knowledge (e.g., high-level domain concepts).

The results show that: 1) logs are a valuable source of both development and field knowledge; 2) the logs are continuously evolving; and 3) developers often change logs without considering the dependence of operators on them, leading to very fragile *Log Processing Apps*. The results of this study show that more systematic approaches are needed to leverage the rich development and field knowledge in logs.

IV. Log Analysis to Support Developers and Operators

This section presents our proposed approaches based on logs to support software developers and operators. For each approach, we present the problem that we wish to solve, the reason why development or field knowledge in logs can help, and a brief overview of our proposed approach.

A. Leveraging Field Knowledge to Improve the Quality of Software for Developers

Identifying Error-Prone Software Components Using Log Maintenance History

Problem: Software components that are vulnerable to field errors are often hard to identify using traditional pre-deployment testing or static analysis.

How field knowledge can help: In an effort to track and diagnose such errors, operators often require changes or additions to logs. Thus, we believe that frequent updates to logs (i.e., high log churn) in particular components are good signs of problems and future bugs.

Our proposed approach: Using the development history, we will measure the amount of log churn for every software component (e.g., class). We will then build statistical models of post-release bugs for every component using the log churn information and other traditional metrics, such as pre-release bugs, total number of changes, and code complexity. We will study the statistical model to understand the relation between log churn and post-release bugs. We expect that log churn will be statistically significant in explaining post-release bugs and can complement traditional metrics. We will perform case studies on several open source and commercial systems to evaluate our approach.

Evaluating Field-Test Coverage for Large-Scale Systems

Problem: Testing of large-scale software system aims to test all its functionalities. In practice, software testers typically run software systems for a long time period to achieve high test coverage. However, the high test coverage still does not guarantee a reliable system in the field. It is important for software testers to understand the coverage for a system test based on field data.

How field knowledge can help: To determine the quality of the test coverage, systems are often instrumented. However such instrumentation is not feasible for field deployment. Therefore, often such type of field-test coverage analysis is not performed. We believe that we can leverage logs to give us an approximation of instrumentations with no additional performance impact. Software testers can then evaluate test coverage by comparing logs from the test-runs and field deployments. Our hypothesis is that the logs fro test-runs can be used to evaluate the field-test coverage.

Our proposed approach: The first step of our approach is to create field execution models from the field logs. A log line typically contains several parameter values, such as time stamp and session ID. We will group log lines that have the same parameter value together into log sequences. Each log sequence corresponds to a step-by-step high-level summary of the run-time behaviour of the systems. We will use such log sequences as our field execution model. In the second step, we will use the same approach on testing logs to derive a testing execution model. In the third step, we will compare both models to find out how much of the field execution model is covered by the testing execution

model. We plan to use approaches similar to the reflexion model, proposed by Murphy *et at*. [5]. We will evaluate our approach against extensive instrumentation to study whether the two test coverage measurements are similar.

B. Leveraging Development Knowledge to Cope with the Operational Complexity of Software for Operators

Automated Documentation of Log Lines

Problem: Other than the source code, there is no explicit documentation for logs in current software engineering practice. Furthermore, there is only limited communication between developers and operators. However, ecosystems of *Log Processing Apps* that depend on the logs have to be built based on the operators' assumptions on logs. Therefore, fully understanding the rationale behind log lines is important to ensure the correctness of such *Log Processing Apps* and to assist operators in rapid diagnosis of field problems.

How development knowledge can help: The source code context of a log line is rarely known by operators, yet that context is recorded by developers in their changes to the source code. The historical knowledge about these changes can be used as a way to document and explain log lines.

Our proposed approach: Our approach is to attach the development history as well as bug reports to every log line. To achieve this, we need to link source code with log lines first. First, we adopt an approach proposed by Xu *et al*. [6] to generate templates for logs (e.g., a regular expression of log format) using static analysis on source code. Each source code unit, i.e., method or class, now has a list of log templates. We will match the log templates with the actual logs to link every log line with the source code that generates it. Therefore, together with the source code, the entire software development knowledge, such as the history of the source code and the bug reports, can all be linked with the logs, similar to the sticky note approach by Hassan and Holt [7]. To evaluate our approach, we will perform qualitative studies and present the attached development history to domain experts of several systems.

Log Reduction Using Development History

Problem: When a new software release is deployed in the field, operators need to examine all the logs to understand the impact of the new release. However, typically not all the functionality is affected in the new release, leading to a waste of effort.

How development knowledge can help: The development knowledge tells which source code snippets are changed. We can derive the affected logs from the changed source code. We can then filter the generated logs to only show logs that are likely to be impacted by recent changes instead of showing all generated logs. Our hypothesis is that the recent changed logs are more critical to operators.

Our proposed approach: We will analyze the source code changed in a new release and we identify the log templates that are possibly impacted. Such impact includes direct impact (e.g., the method that generates the log line is modified) and indirect impact (e.g., the method generating logs may be invoked by the changed method). We use the information to create log filters that are based on the development history. We will apply our approach on open source and commercial systems and perform user studies with operators to evaluate the usefulness of our approach.

V. Log Analysis Challenges

In this section, we present the challenges of log analysis. For each of the challenges, we present the challenge and our proposed solution to address the challenge.

C1. Logs are Non-structured.

Logs are generated by the statements inserted by developers to record valuable information about system execution. Developers typically do not follow a certain format to generate logs. Even if there is a format, developers do not follow the format consistently [8].

We address the non-structured nature of logs by transforming logs into a structured data model. Jiang *et al*. [9] propose that execution logs can be abstracted into static execution events and a list of dynamic parameter values. For example, a log line "time=1, Trying to launch, TaskID=01A" can be abstracted into an execution event "Trying to launch" and two parameter values, "1" and "01A". Based on this existing log abstraction technique, we further generalize logs into relations. The above example log line can be transformed into a relation with 3 attributes: execution event, time and TaskID. This particular tuple in the relation has the value "(Trying to launch, 1, 01A)". Therefore, a log corpus can be structured as a list of relations.

C2. The Processes for Logging and the Use of Logs are Largely Ad Hoc.

Developers typically do not realize the extensive use of logs by others. Operators typically have to use the logs in an ad hoc fashion using Perl or shell scripts. Using such scripts is not efficient because they are ad hoc and they can hardly be adopted to solve other problems.

Relational algebra has been applied on software development data [10]. The similarity between logs and software development data leads us to exploit relational algebra [11].

C3. The Large Scale of Logs.

One of the major challenges of log analysis is the large scale of data. For example, a small testbed setup of a typical *Hadoop* cluster already generates over 200 MB of logs per second [12].

We propose to implement a query and manipulation language on top of a web mining framework called *Pig* [3]. We have already started implementing the approaches presented in Section IV using this language. In our previous research, we used the web mining framework to enable large-scale Mining Software Repository studies [13].

VI. State of the Art and Practice

The following lines of research are closely related to the work presented in this thesis.

Log Analysis Techniques to Assist in Development

Researchers have proposed various automated log analysis techniques to assist in development. Jiang *et al*. [14] design log analysis techniques to assist in functional verification of software load tests. Beschastnikh *et al*. [15] develop an automated tool that infers execution models from logs. The models can be used by developers to diagnose bugs. Yuan *et al*. [16] propose an approach to improve diagnosing problems in software systems by enhancing the logging code of the system. Nagappan *et al*. [17] design an efficient algorithm to extract operational profiles from logs.

Our proposed approaches are similar to these techniques. However, we aim to ensure that our approaches are scalable for very large-scale systems. We will be working with an industrial partner who will provide us access to a very large corpus of logs.

Scalable and Systematic Log Analysis Platforms

Splunk [18] is a semi-structured time series database frequently used today in practice as a log-processing platform. Splunk indexes log data and supports scalable searching for keywords in logs. Similarly, IBM InfoSphere Streams [19] is designed to store, index and analyze massive volumes of data at rates up to petabytes per day continuously. Software execution logs are one of the major targets of IBM InfoSphere Streams.

These existing platforms only support simple grep-like operations but are not able to perform complex analyses. Our work attempts to create a general log analysis framework using a relational algebra formalism.

VII. Conclusion

Our pilot empirical study on large software systems highlights the growing gap between software developers and operators during software maintenance activities. This thesis contributes to the software engineering and system community by proposing to leverage logs to bridge the gap. To benefit developers, we will use the field knowledge in logs to identify error-prone software components and evaluate test coverage. To benefit operators, we will attach the development history to logs as a way to automate the documentation of logs and to support log reduction. To overcome the challenges of log analysis, we will develop a relational-algebra based language as our infrastructure.

Acknowledgment

The author thanks Dr. Ahmed E. Hassan and Dr. Bram Adams for their comments on earlier drafts. The author appreciates the generosity of the Performance Engineering team at Research In Motion (RIM). Working with the team as an embedded researcher, the author has gained an appreciation of the current practice and the daily challenges of mining large-scale software execution logs.

References

[1] "Devops," http://www.devopsdays.org/.

[2] "Summary of sarbanes-oxley act," http://www.soxlaw.com/.

[3] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *SIGMOD '08: Proc. of the 2008 ACM SIGMOD Int. Conf. on Management of data*, pp. 1099–1110.

[4] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. Godfrey, M. Nasser, and P. Flora, "An Exploratory Study of the Evolution of Communicated Information about the Execution of Large Software Systems," in *WCRE '11: Proc. of the 18th Working Conf. on Reverse Eng.*, October, pp. 335 –344.

[5] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: bridging the gap between source and high-level models," in *FSE '95: Proc. of the 3rd ACM SIGSOFT Symp. on Foundations of software engineering*, pp. 18–28.

[6] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP '09: Proc. of the ACM SIGOPS 22nd symp. on Operating systems principles*, 2009, pp. 117–132.

[7] A. Hassan and R. Holt, "Using development history sticky notes to understand software architecture," in *IWPC '04: Proc. of 12th IEEE Int. Workshop on Program Comprehension*, pp. 183 – 192.

[8] M. Bruntink, A. van Deursen, M. D'Hondt, and T. Tourwé, "Simple crosscutting concerns are not so simple: analysing variability in large-scale idioms-based implementations," in *AOSD '07: Proc. of the 6th Int. Conf. on Aspect-oriented software development*, pp. 199–211.

[9] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *J. Softw. Maint. Evol.*, vol. 20, no. 4, pp. 249–267, 2008.

[10] R. C. Holt, "WCRE 1998 Most Influential Paper: Grokking Software Architecture," in *WCRE '08: Proc. of the 2008 15th Working Conf. on Reverse Engineering*, pp. 5–14.

[11] A. Tarski, "On the calculus of relations," *The J. of Symb. Logic*, vol. 6, no. 3, pp. 73–89, 1941.

[12] A. Rabkin and R. Katz, "Chukwa: a system for reliable large-scale log collection," in *LISA'10: Proc. of the 24th Int. Conf. on Large installation system administration*, pp. 1–15.

[13] W. Shang, B. Adams, and A. E. Hassan, "Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report," *J. of Sys. and Softw,*, 2011, in Press.

[14] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *ICSM '08: Proc. of 24th IEEE Int. Conf. on Softw. Maint.*, pp. 307–316.

[15] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *ESEC/FSE '11: Proc. of the 19th ACM SIGSOFT Symp. and the 13th Euro. Conf. on Foundations of software engineering*, pp. 267–277.

[16] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," in *ASPLOS '11: Proc. of the 16th Int. Conf. on Arch. support for programming languages and oper. sys.*, pp. 3–14.

[17] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *ISSRE '09: Proc. of the 2009 20th Int. Symp. on Software Reliability Engineering*, pp. 41–50.

[18] L. Bitincka, A. Ganapathi, S. Sorkin, and S. Zhang, "Optimizing data analysis with a semi-structured time series database," in *SLAML'10: Proc. of the 2010 workshop on Managing sys. via log analysis and machine learning tech.*, pp. 7–7.

[19] "Infosphere streams," http://goo.gl/nI1a4.