Can You Capture Information As You Intend To? A Case Study on Logging Practice in Industry

Guoping Rong†, Yangchen Xu‡, Shenghui Gu‡, He Zhang†, Dong Shao†
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University, Nanjing, China
†{ronggp, hezhang, dongshao}@nju.edu.cn, ‡{mf1932211, dz1732002}@smail.nju.edu.cn

***Abstract*—*Background:* Logs provide crucial information to understand the dynamic behavior of software systems in modern software development and maintenance. Usually, logs are produced by log statements which will be triggered and executed under certain conditions. However, current studies paid very limited attention to developers' *Intentions* and *Concerns* (*I&C*) on logging practice, leading uncertainty that whether the developers' *I&C* are properly reflected by log statements and questionable capability to capture the expected information of system behaviors in logs. *Objective:* This study aims to reveal the status of developers' *I&C* on logging practice and more importantly, how the *I&C* are properly reflected in software source code in real-world software development. *Method:* We collected evidence from two sources of a series of interviews and source code analysis which are conducted in a big-data company, followed by consolidation and analysis of the evidence. *Results:* Major gaps and inconsistencies have been identified between the developers' *I&C* and real log statements in source code. Many code snippets contained no log statements that the interviewees claimed to have inserted. *Conclusion:* Developers' original *I&C* towards logging practice are usually poorly realized, which inevitably impacted the motivation and purpose to conduct this practice.**

***Index Terms*—logging practice, developer, intentions and concerns, inconsistencies**

I. Introduction

Logging practice is a widely used technical approach in modern software development and maintenance. In general, logging practice can record the program's behavior during execution and thus provide valuable information (usually contained in log files) for quality management practices later on, e.g., program analyzing, debugging, and so on. On some occasions, such information is the only data source available in production environments [1]–[3]. Due to this feature, logging practice attracts a lot of attention among software engineering practitioners and researchers, resulting in many advanced logging methods, frameworks, and tools. Nevertheless, a noteworthy fact is that the current mainstream logging practice still relies mainly on the developers to manually inject log statements at the appropriate places in the source code. Therefore, the developers' personal expertise and experiences play a vital role to determine the place (*where to log?*) and the content (*what to log?*) of a log statement. To facilitate understanding, we clarify the terminology and some relevant concepts around logging practice in Table I.

Normally, developers should write log statements with specific *Intentions*, e.g., to capture a performance anomaly or just plain errors. Meanwhile, logging carries certain costs such as performance decline, security risks, etc. Therefore, developers should also have specific *Concerns* regarding logging practice. Typically, developers should make careful considerations on both *I&C* (abbr. for *Intentions* and *Concerns*). Therefore, the developer's *I&C* are valuable, even essential, for understanding and improving logging practice. If we only focus on the two-W questions (i.e., *what to log?* and *where to log?*) and ignore the developer's corresponding *I&C*, it is easy to fall into the dilemma of "know the hows but not the whys". To this end, we carried out this study with the aim to understand developers' *I&C* when performing logging practice and more importantly, we go one step further to investigate how the *I&C* are properly reflected in the artifacts (i.e., the source code).

To address this research purpose, we conducted a case study at *XH*, a world-leading company on big-data technology. We collected evidence from two sources, i.e., the interviews with developers and the real log statements in the source code they developed, respectively. With mutual comparison on the two sources of evidence, we attempted to shed light on the developers' *I&C* and identify potential inconsistencies between the developers' *I&C* and the eventual implementation of the logging practice (i.e., the real log statements in source code). Results suggest that the developers' *I&C* have been poorly implemented in logging practice, which to a certain degree confirms the observations and explains the reason as well in several studies [1], [4]–[6].

TABLE I
The Terminology Adopted in This Paper

Log	A log is a collection of the execution outputs of log statements, which is usually stored in a textual file or a database.
Log Statement	A log statement is a statement placed in the source code that can output a record of the behavior of a given program during its execution.
Logging Practice	Logging practice refers to multiple practical tasks in logging, from inserting log statements into source code, to using the information provided by generated logs to help program debugging and analyzing. In general, logging practice includes log placement and log analysis.
Log Placement	Log placement is a part of logging practice that refers to decisions made by developers on the places (*where to log?*) and contents (*what to log?*) of log statements in source code.
Log Analysis	Log analysis is another part of logging practice that makes use of generated logs to achieve analysis tasks such as error debugging, performance analysis, system behavior understanding, etc.
Log Level	Log level reflects the severity of the log message. It allows developers to specify the appropriate amount of logs to print during the execution of the software. For example, in *Log4j*, there are seven levels (ordered by severity): ALL, DEBUG, INFO, WARN, ERROR, FATAL, and OFF.

2576-3148/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSME46990.2020.00012

The main contributions of this study can be highlighted as follows:

- It focuses on the logging practice in industry and sheds light on the current state of logging practice adopted in industry, which is rare in the previous studies.
- It applies a mutual verification on the evidence collected from two sources (i.e., interview and corresponding real log statements) and a further confirmation to reveal the developers' *I&C* towards logging practice and whether the *I&C* are properly reflected in software source code, to which current studies paid very limited attention.
- It identifies major inconsistencies between the developers' *I&C* and their implementation in source code, which not only reveals challenges but also implies improvement directions.

The rest of this paper is organized as the following. In Section II, we discuss the related work about logging practice. Section III introduces our study methodology, including design and execution. In Section IV, we elaborate the results and findings. In Section V, we discuss the implication of the findings and possible solutions. Threats to validity are discussed in Section VI. Finally, conclusions and future work are presented in Section VII.

II. Related Work

Because the log information has been considered to be the only means to provide detailed information that captures the dynamic behavior of the running program [2], [7], [8], research on logging practice has attracted more and more attention. Nevertheless, how this practice has been adopted in the industry is not clear due to the relatively smaller amount of studies based on the industrial context. In this section, we introduce some related work about the research on logging practice.

A. Logging Practice

Around logging practice, there exist several categories of research work as follows:

Where to log? These studies concern the location that log statements should be placed. For example, Fu et al. [4] conducted an empirical study in Microsoft to study the developers' logging practice with regard to *"where to log"*. Further, for the purpose of facilitating usage, prototype tools are always raised along with the logging practice approach. For example, Zhu et al. [8] developed a tool named *LogAdvisor* that suggests developers whether they should place a log statement in a code snippet. Yuan et al. [9] developed *Errlog* that adds proactive log statements into source code without introducing heavy logging overhead. The common idea of these tools is to provide suggestions on whether to place log statements in a code snippet by learning or summarizing log patterns derived from existing log statements in source code. More specifically, several studies focus on the log-placement algorithm. For example, Zhao et al. [10] proposed an algorithm that computes the "best" placement of INFO-log statements with a pre-defined performance overhead threshold.

What to log? This type of studies usually focuses on the contents of the log statements. For example, Yuan et al. [11] presented a tool, *LogEnhancer* that systematically enhances every log statement in source code to collect causally related diagnostic information. Li et al. [12] proposed an automated approach to help developers determining the most appropriate logging level when adding a log statement to the source code. Nevertheless, an empirical study involving more than 1000 open source GitHub repositories conducted by OverOps[1] implies that most log statements do not contain any parameter, rendering questionable capability to capture the runtime behavior of the target software system via logging practice.

Logging patterns or characteristics. Many studies cover both aspects (i.e., *where to log?* and *what to log?*) around logging practice. The basic idea is by studying existing log placement in many projects so that to expect certain logging patterns or characteristics, which may help developers to do better logging practice. For example, Yuan et al. [1], [9] studied the logging practice of open-source software projects and further proposed proactive logging strategies. Similarly, other studies such as log change behaviors [1], [13], [14], log evolution [15], and anti-patterns in log statements [16] also try to provide reference value for developers when conducting logging practice.

All the aforementioned studies provide insightful findings on logging practice from various perspectives. However, there seems not enough attention has been paid on the *I&C* of developers while they carrying out logging practice, given the fact that most of the log statements are manually written by the developers. Some studies claimed to investigate "logging intention" and its value for log placement improvement [17] and log level recommendation [18]. Nevertheless, the "logging intention"s in these studies are derived from the researchers' perception and conjecture based on the contextual features of log statements in the source code, which never has been confirmed by the developers who produced them. In fact, as study [19] implies, it is not rare that developers may simply copy log statements and paste them to different code snippets without any modification. As a result, there are a lot of duplicated log statements in these projects, which easily mislead researchers' perception and conjecture on the real logging *I&C* of developers without necessary confirmation from the person who wrote the log statements. In this study, we value the importance of the developers' real *I&C* for better logging practice in future development and apply face-to-face interviews together with source code analysis to collect evidence to reveal the developers' real *I&C* towards logging practice.

B. Study on Logging Practice in Industry

Studies on logging practice in industrial companies are relatively rare, compared to the large number of studies using open source projects. One reason for this phenomenon might

[1]https://land.overops.com/java-logging-in-production-ebook/

be that it is much easier to obtain the source code and log statements in open source projects. Nevertheless, in open-source projects, it is also relatively difficult to get access to the developers who wrote the log statements and learn about their *I&C* when they put the log statements in the source code. In industrial companies, the situation is exactly the opposite. Since it is usually easier to find the developers, with interviews, their *I&C* during logging are thus easier to be acquired, although obtaining source code might be a challenge.

Baccanico et al. [20] interviewed several developers from different teams in one company, where they developed, integrated and maintained distinct products from a given product family. It turned out that developers even within a given product family rarely share rules about message structure and logging architecture. The authors believed that this was a common drawback and a known issue for any large-scale development. Pecchia et al. [5] also interviewed the developers in an industrial organization. They observed that the developers in the same product line share common rules regarding the implementation of the log statements. Nevertheless, there were no strict and explicit logging rules across different product lines. As a result, in this organization, three product lines have different log structures and different log production mechanisms. Apparently, the *I&C* of developers are still missing from these studies. Fu et al. carried out a study containing code analysis and a questionnaire-based survey at Microsoft [4] to explore the logging decisions regarding *"where to log?"*. Nevertheless, we intend to go one step further to understand the *I&C* behind these decisions and possible gaps between the *I&C* and actual log placement.

III. Research Method

We value the determinant role of developers' subjective opinions (i.e., *I&C*) in current logging practice. Therefore, our research attempts to establish proper understanding of developers' *I&C* and the degree to which they were properly reflected in the log statements in source code in real-world software development. In this section, we elaborate on our research method.

A. Background

Our research was conducted at *XH*, a world-leading IT company, focusing on enterprise-level cloud computing, big data, and AI technologies. At present, XH holds approximately 10 enterprise-level products and houses over 600 employees, among which 20% of employees work in the R&D department. The considerations are two-fold. First, as studies imply, platform providers tended to rely on logging mechanism for issue diagnosis [1], [21]. Second, as long as we are also interested in the status of *I&C* being reflected in log placement, the source code in companies with strict regulations on logging practice(e.g. [22]) may not be able to reflect developers' *I&C* freely in log placement.

In our research, we selected three Java-based projects covering three major product lines in *XH*, i.e., storage engine, computing engine, and middle-ware, respectively. The following is a brief introduction to them.

- **HD** is a flash-based distributed storage engine that ensures the performance, stability, and reliability of big data platforms.
- **IC** is a high-performance distributed computing engine for data marts and real-time data warehouse.
- **SG** is a middle layer software between a general distributed data service layer and the underlying storage engine, which further abstracts the underlying storage engine into a set of interfaces.

The overview information of the three projects is shown in Table II. The biggest project is *IC*, which is developed and maintained by 7 engineers. The *IC* project contains $133,132$ lines of code, among which there are $1,402$ lines of log statements.

TABLE II
Overview of the Three Projects: lines of code, lines of log statement and the number of developers

Project	Lines Of Code	Lines Of Log Statement	# Developers
HD	61,567	598	4
IC	133,132	1,402	7
SG	71,345	195	4

B. Research Design

1) Research Questions: To be specific, we describe the goal of this study using a GQM style [23] as the following:

To investigate and analyze *current logging practice (more specifically, the log placement, which includes both "where to log?" and "what to log?")*

For the purpose of *understanding the developer's I&C and identify potential inconsistencies between the I&C and the log statements in source code*

From the perspective of *developers' I&C*

In the context of *real-world software projects in one big-data company.*

To achieve the research goal, we have defined the following four research questions:

RQ1*: What are the developers' awareness and understanding of logging practice in this company?*

Developers' awareness and understanding of logging practice may first have an impact on the practice. For instance, whether they are aware of the importance of logging? To what degree developers accept logging practice as a common task during development? etc. Obviously, without a shared awareness and understanding of logging practice, we may not be able to expect proper log placement in the source code.

RQ2*: What I&C that developers have when they conducting the logging practice?*

The developers' *I&C* are intangible yet extremely critical in logging practice. The importance of this research question is that the developers' *I&C* are supposed to be reflected in log

placement. In short, the developers' *I&C* drive the logging practice they carried out, which may provide clues for us to understand the log placement we identified in the source code. Meanwhile, it may also provide a comparison basis to identify possible inconsistencies between what the developers thought (i.e., their *I&C*) and what they did (i.e., the real log statements in source code) with respect to the logging practice, which leads to the next research question.

***RQ3**: To what extent the developers' I&C could be properly reflected in the source code regarding log placement?*

Ideally, the *I&C* described by the developers should be consistent with the log placement in the program. However, for example, compared to the business logic code, log statements are very easy to be neglected, especially in development under time pressure. Moreover, the source code might also change for various reasons (e.g., to fix a defect or add a new feature), rendering more chances for inconsistency between the *I&C* and log placement. In this sense, we define this research question to understand the status that whether the log placement is able to reflect developers' *I&C* consistently.

***RQ4**: What should be improved regarding the current adoption status of logging practice?*

If current logging practice can not support the developers' *I&C* to be properly reflected, we intend to explore improvement opportunities from the perspective of the forefront engineers.

2) Research Process: In order to avoid any distraction, i.e., we do not want interviewees to explain existing source code containing log statements, we first perform **Interviews**, during which interviewees answer questions without referring to the actual source code they wrote. Then an independent **Code analysis** will be conducted to provide objective evidence regarding the log placement in the actual source code. By comparing the results from both sources and a confirmation interview to discuss the preliminary findings and the reasons behind in a **Result synthesis** step, we may be able to understand the status that how the *I&C* are properly implemented in source code. Figure 1 depicts the research process of our study.

Step 1: Interview.

We first prepared a set of questions as shown in Table III, driven by the research goals and questions. A mapping between the interview questions and research questions is also presented in Table III. The questions covered topics such as the interviewee's background, current logging practice, and expectations for better logging practice. Then we conducted a series of interviews with the developers.

Step 2: Code analysis.

After the interviews, we started source code analysis. Log statements were extracted with key relevant information (e.g., file path, contributor, etc.) from the source code. Through manual check of each log statement, we intended to collect contextual features (e.g., the code structure, the parameters in log statements, etc.) of the code snippet (typically, a method or major branch of a big method) containing log statements.

Step 3: Result synthesis.

With the results from both the previous steps (i.e., Step 1 and 2), we were able to conduct a mutual verification by comparing the two sources of evidence, through which we attempted to examine how the developers' *I&C* are reflected in the real log statements. To increase the credibility of the results and, more importantly, to explore the reasons behind the results we obtained, we finally conducted the follow-up interviews as the confirmation with the interviewees in this step.

C. Execution

1) Interviews and Data collection: The interview session was conducted at *XH*. Two full-time members (one team leader and one core developer) were invited to participate in the interviews. As a result, we had six interviewees for the interview session. To better collect information, we conducted all the interviews with only one developer at a time. All the researchers attended these interviews with different roles. One researcher asked questions and all the other researchers took the role of recorder and took notes. If necessary, the recorders can also ask extra questions to obtain more information from the interviewees. The person who asked questions also acted as the timekeeper to limit the duration of each interview within 60 minutes. Table IV lists the duration of each interview, which ranges from 32 minutes to 45 minutes.

2) Tool extraction, manual check, and data collection: We developed a tool to manage this step, which encapsulates several shell and git commands (as shown in Code 1) to generate Git blame files and extract important information such as the path of source code file, the author, the line number and original statement of each line of code, etc. To secure the quality, we manually examined and double-checked the extracted log statements for subsequent analysis.

```
# generate Git blame files, which contain the path, author, line number and original statement of each line of code
git ls-files | grep -E "\.java$" | sed 's/\(.*\.\)java$/git blame \1java > \1blame/' | sh

# use regular expressions to collect log statements
find . -name '*.blame' | xargs grep -iE \
   -e 'log[^. ]*\.[^.]*(fatal|error|warn|info|debug|trace)[^. ]*' \
   -e 'log[^. ]*\.log\((severe|warn|info|config|fine|finer|finest)' \
   -e 'log[^. ]*\.log\(level\.(severe|warn|info|config|fine|finer|finest)'
```

Code 1. Example commands for log statement extraction

3) Evidence analysis and synthesis: With evidence collected from two different sources (i.e., interview and code analysis), it is possible to mutually compare and verify the evidence and identify whether there exists inconsistency between two different sources of evidence, in other words, whether *what they did* was consistent with *what they said*. We did this with pairs of researchers so as to avoid omission and misconception of the above-mentioned inconsistency. Since most questions in Table III are open questions, resulting in massive irrelevant information in the notes researchers

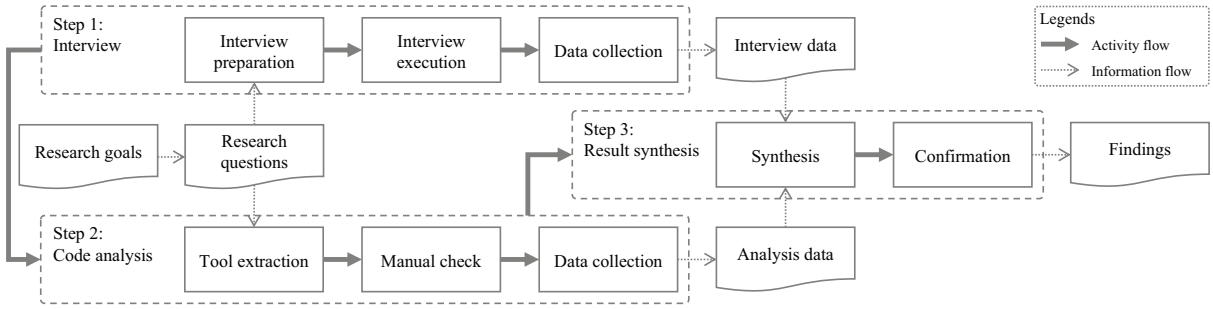

Fig. 1. The Process of This Research

TABLE III
THE INTERVIEW SCRIPT USED IN THIS STUDY

Section	Interview Questions	Question Content	To Answer RQ(s)
Background	IVQ1	How many years of working experience do you have as a developer?	RQ1
	IVQ2	How many years have you been working in current company?	RQ1
	IVQ3	Do you use logging tools? Which logging tools have you ever used?	RQ1
	IVQ4	What kind of development activities have you ever participated in related to logging practice?	RQ1
Current Practice	IVQ5	Usually, why do you insert log statements in the source code you wrote?	RQ2
	IVQ6	Where do you usually add log statements in the source code you wrote?	RQ2
	IVQ7	Do you have any concerns when you inserting log statements?	RQ2
	IVQ8	Is there any overhead you would concern when putting log statements?	RQ2
	IVQ9	Continue with IVQ8, what types of overhead you usually care about in logging practice?	RQ2
	IVQ10	To what degree, your logging intentions could be supported by current logging practice? and why?	RQ3
	IVQ11	Is there any guidelines supporting your current logging practice?	RQ4
	IVQ12	Do you and how often do you modify your log statements?	RQ4
	IVQ13	Can you introduce us some reasons or examples that you have to modify your log statements?	RQ4
	IVQ14	Usually, do you consider modifying the log statements when the corresponding business code changed?	RQ4
	IVQ15	Given the current status of logging practice in your project, what difficulties or challenges have you encountered in current logging practice?	RQ4
	IVQ16	Regarding the logging tools you adopted in your projects, what are the major problems or limitations?	RQ4
Improvement Expectations	IVQ17	Do you think logging guidelines will be helpful? and what kind of the information should be contained in the logging guidelines?	RQ1/RQ4
	IVQ18	What kind of logging tools do you think will be helpful? For example, what features should be included?	RQ1/RQ4
	IVQ19	What do you think need to or must be improved in current logging practice?	RQ4

TABLE IV
INTERVIEW DURATION

	HD1	HD2	IC1	IC2	SG1	SG2
Duration (min)	41	35	36	43	45	32

recorded. To maintain focus on relevant information, we also devised several questions to guide us throughout the evidence consolidation and analysis.

(1) *What are the typical places (both oral evidence and artifact-based evidence) where developers tend to put log statements?*
(2) *Did the developers put log statements at typical places as they said?*
(3) *Did the developers put log statements at places where they claimed not necessary?*
(4) *Did the developers write log statements with suitable log levels?*
(5) *Did the developers write log statements for the purpose of performance diagnosis? etc.*

4) Confirmation interview: Based on the results from the above analysis, we identified several types of log-prone code snippets (see Table VII). We then collected code snippets that were added or modified within the last month, with the consideration that the developers may still have a good memory of code snippets so that they still remember their *I&C* to the log statements. We still interviewed one developer at a time. For each interview, we discussed the reason behind each code snippet regarding logging practice, no matter whether there exists a log statement or not. The confirmation interviews provide valuable information for us to better understand the developers' *I&C* and the inconsistencies between the developers' *I&C* and actual log placement.

IV. RESULTS & FINDINGS

We discuss results and findings in this section.

A. RQ1: Developers' awareness and understanding

Table V presents the background of our interviewees. Although the interviewees have various development experience and work experience in *XH*, their participation in activities around logging practice is similar, as all the interviewees have ever "used logging tools", "configured logging tools" (e.g., to write *log4j.properties*) and "performed log analysis". In fact, this is also reflected in the answers to *IVQ3*, as they all have experience in using logging frameworks and tools such as *Log4j* and *Slf4j*. Besides, two interviewees (i.e., HD1 and IC1) with relatively more development experience even "designed

TABLE V
BACKGROUND OF THE INTERVIEWEES

Interviewee	Participation in Activities Around Logging Practice			
	Using logging tools	Configuring logging tools	Designing logg -ing interfaces	Performing log analysis
HD1(8,4)*	✓	✓	✓	✓
HD2(4,2)	✓	✓	-	✓
IC1(10,4)	✓	✓	✓	✓
IC2(5,2)	✓	✓	-	✓
SG1(8,4)	✓	✓	-	✓
SG2(3,2)	✓	✓	-	✓

* HD1(8,4) means the first interviewee from project *HD* has 8 years of experience on software development and 4 years of experience on current job, and so on.

TABLE VI
INTERVIEWEES' MOST MENTIONED I&C TOWARDS LOGGING PRACTICE

Answer \ Interviewee	HD1	HD2	IC1	IC2	SG1	SG2
Intentions						
Error debugging	✓	✓	✓	✓	✓	✓
System behavior understanding	✓	✓	✓	✓	-	-
Performance diagnosis	-	-	✓	✓	-	-
Concerned Overhead						
I/O	✓	✓	✓	✓	✓	✓
Memory	✓	✓	✓	✓	✓	✓
CPU	-	-	✓	✓	-	-
Storage	-	-	✓	✓	-	-

logging interfaces". For example, sometimes existing logging frameworks and tools could not satisfy the advanced logging requirements (e.g., logging performance info on a sequence of business functionalities) during project development. In this case, sophisticated developers would design a project-specific logging interface to make restrictions on the log statements through a set of logging rules, as well as facilitate common usage of existing logging frameworks and tools.

While talking about logging, the interviewees also shared opinions like "*Logging is very helpful for failure diagnosis. It is now an integral part of my daily development work.*" and "*I have different concerns when writing log statements, since I think that logging varies in different places and different situations.*" during the interviews, which also to a certain degree reflect that logging practice is a common part in daily development for the interviewees.

Finding 1: We have not observed developers with no experience on logging practice, which to a certain degree implies the practice is a common task in software development in *XH*. Meanwhile, current logging tools may not support some advanced logging requirements well for sophisticated developers.

B. RQ2: Developers' I&C

Through *IVQ5*, we tried to obtain the interviewees' *Intentions* to put log statements in source code. As shown in Table VI, the interviewees mentioned "error debugging", "system behavior understanding" and "performance diagnosis" as the logging intentions. A noteworthy point is that only the interviewees from project *IC* mentioned to write log statements for "performance diagnosis", which can be verified in the source code for this project.

TABLE VII
CODE SNIPPETS TO PUT LOG STATEMENTS

Code Snippet & Elaboration	Interviewees Mentioned
Exception Handling Developers put log statements in an exception handling block (e.g., try-catch block) to record exceptions, i.e., the abnormal behavior of the program.	HD1, HD2, IC1, IC2, SG1, SG2
Condition Check Developers put log statements in a condition check block (e.g., if-else block) to capture different program behaviors under different conditions.	HD1, HD2, IC1, IC2, SG1, SG2
Interface Invocation Developers put log statements right after an interface invocation to make sure that the interface invocation returns the right result.	HD1, HD2, IC1, IC2
Hotspot Developers put log statements in the code snippets that are "believed to be" error-prone due to complex logic or that own key responsibilities for business (which we called *hotspot*) to improve the maintainability of the system.	HD1, HD2, IC1, IC2, SG1, SG2

Moreover, while we were synthesizing the interviewees' answers to *IVQ6* (as shown in Table VII), we noticed that different *Intentions* may lead to different places where developers put log statements. This result was verified in the source code analysis. For example, log statements with the intention of "error debugging" can usually be found in "exception handling" or "condition check" code snippets with parameters to record the values of certain variables. Meanwhile, log statements with the intention for "performance diagnosis" are usually put in some "hotspot" code snippets, recording the execution times of certain functions.

Logging costs resources, therefore, developers need to seek a balance between the cost and benefit, in short, the *Concerns*. We summarize three major concerns mentioned by the interviewees as the following.

Log content. The interviewees believed that the information being captured (i.e., log content such as parameters) through logging was one of the most considered when they wrote the log statements.

Log level. What is a proper log level for a particular log statement in a specific code snippet is one of the most mentioned concerns. Moreover, determining a suitable log level is also the most common difficulty faced by the interviewees (ref. Section IV-D).

Log overhead. As shown in Table VI, most interviewees mentioned the overhead of logging as one of the major concerns, with "I/O" and "memory" being the most mentioned. Two interviewees (i.e., IC1 and IC2) also mentioned two more types of overhead: "CPU" and "storage". We believe that different project characteristics (type, objective, etc.) may lead to different concerns on the log overhead from developers. For example, project *IC* is a high-performance computing engine, so the logging should be conducted with cautions on nearly all aspects related to the system performance, given the fact that "not logging, no extra cost".

Finding 2: Project characteristics (e.g., whether performance is critical) may impact the developers' *I&C*, leading to

different log placement potentially. Nevertheless, developers in different projects may still share some *I&C* in common.

C. RQ3: Log placement to reflect developers' I&C

Since we only attempt to uncover the potential gap between the *I&C* and the artifacts, we only involve one typical inconsistency in our study, i.e., the source code snippet with absent log statement that the owner (interviewee) claimed to have. In fact, this is also one of the most occurred issues on logging practice, as studies [6], [11], [24] indicated.

Code 2 shows an example of "absent logging" inconsistency. According to Table VII, both "condition check" and "interface invocation" should contain log statements since these places are error-prone places due to various unexpected reasons. *Line* 3 is a "condition check" and there does exist a log statement at *line* 4, whereas there is no log statement after the "interface invocation" at *line* 9. Therefore, Code 2 is taken as one inconsistency.

```
1  public void storeFile(File file) {
2      ...
3      if (!file.exists()) {
4          logger.error("File do not exists!");
5          ...
6      }
7      ...
8      UUID uuid = UUID.randomUUID().toString().replace("-",
           ↪ "");
9      Path storagePath = StorageService.generateStoragePath
           ↪ (file, uuid);
10     Files.copy(file, path, StandardCopyOption.ATOMIC_MOVE
           ↪ );
11     ...
12 }
```

Code 2. An example of an "absent logging" inconsistency

```
1  public void create(Handler handler, Table table) {
2      ...
3      try {
4          ...
5          res = checkFS();
6          if (!res.isValid()) {
7              ...
8              // It should be logged since it is a "condition
                   ↪ check"
9              // However, it is already logged by the catch
                   ↪ block
10             throw new CreateTableException(res.msg);
11         }
12         handler.createTable(table, params);
13         ...
14     } catch (CreateTableException e) {
15         ...
16         logger.error(e.msg, table.schemaName, table.
               ↪ tableName);
17         metastore.rollback(table);
18     } ...
19     } catch (Exception e) {
20       ...
21     }
22 }
```

Code 3. An example of a reasonable "absent logging" code snippet

A noteworthy point is that there exist some exceptional cases. Take Code 3 as an example, there is no log statement after the "condition check" at *line* 6. Nevertheless, an exception is thrown out and then handled by the catch block at *lines* 14-18, where there has a log statement. Thus the absence of the log statement at *lines* 8-10 is reasonable, otherwise the log statement would be redundant. This example shows the necessity that two sources of evidence are required to identify an inconsistency of "absent logging", which also implies exactly a major challenge to establish a practical guideline for logging practice, i.e., the similar context may lead to different decisions on log placement.

TABLE VIII
REFLECTION OF *I&C* IN PROJECTS

Project	Interviewee	# Code Snippets With Absent Log Statements	# Code Snippets Should Contain Log Statements	Percentage
HD	HD1	14	34	41.18%
	HD2	11	25	44.00%
IC	IC1	23	42	54.76%
	IC2	26	44	59.10%
SG	SG1	1	7	14.29%
	SG2	1	5	20.00%

Table VIII portrays the reflection status of developers' *I&C* in the three projects respectively. Apparently, most interviewees fail to put a log statement at every place where they claimed that there should have. For project *HD* and *IC*, a noticeably large part of *I&C* (more than 40%) cannot be reflected in the code. However, project *SG* presents a better status with less than 20% missing logging places. Still, we believe the characteristics of the projects are the reason for this phenomenon.

In the follow-up interview, the inconsistencies have been confirmed and typical root causes for such inconsistencies have been revealed.

Firstly, due to the negligence or carelessness, developers just forgot to insert a log statement at the code snippet where they intended to. This phenomenon reflects a problem regarding logging practices that developers normally are not able to focus on logging practice when coping with source code implementing business logic.

```
1  public void spill() {
2      ...
3      // try {
4      // // check if the server is available.
5      // server.checkAvailable(servername, mode);
6      // ...
7      // } catch (Exception e) {
8      // logger.error("Server cannot be accessed, due to:
           ↪ ", e);
9      // server.recover();
10     // }
11     isAvailable = server.checkAvailable(servername, mode)
           ↪ ;
12     if (isAvailable) {
13         server.addConfig(configuration);
14         server.spill();
15         ...
16     } else {
17         // a log statement should be inserted into this
               ↪ place.
18         server.recover();
19     }
20 }
```

Code 4. An example of a deleted log statement

Secondly, evidence collected from the answers to *IVQ12*, *IVQ13* and *IVQ14* implies that due to various reasons, e.g., new features, interface changes, mistakes, low performance, and business logic changes as well, code modification is

inevitable in daily software development. Nevertheless, log statements are usually neglected in these types of modifications. Thus log statements may be deleted due to changes to source code. Take Code 4 as an example, the original log statement was deleted since the "checkAvailable" interface was changed (from throwing an exception to return a flag). However, after the code changed, no log statement was added into the "else" block to record the state when the server was unavailable.

Finding 3: Developers' *I&C* have not been properly reflected in the source code for different reasons. Exceptional cases exist when a code snippet which seemingly should contain log statements does not actually need to.

D. RQ4: Improvement Opportunities

Evidence from the answers to *IVQ15* and *IVQ16* may help us scratch the surface to understand the major challenges and difficulties that developers may encounter when conducting logging practice. We summarize the challenges and difficulties as follows.

Lack of prompt message regarding log placement. Most interviewees mentioned that sometimes they just missed some important log statements in the source code. One reason for this phenomenon is that the business logic of the source code always attracts the most attention during coding. In this sense, if the IDE or other logging tools could scan the source code and show alerts at the places where there should be log statements, the issue of missing log statements may be addressed to a large degree.

Lack of necessary guidance. Some developers mentioned that the lack of reference guidelines is the reason for relatively poor implementation in the logging practice. Without useful guidelines, log placement is a challenge for them, rendering arbitrary log placement in practice. However, as presented in Table IX, some interviewees do not agree with this viewpoint.

To address the above challenges, some improvement actions have been raised and comprehensively discussed during the interviews. Among them, supporting tools and logging guidelines have been mentioned nearly by all the interviewees.

Tools. Regarding the current logging tools that our interviewees are using, the interviewees mentioned that current tools which only provide a framework to write log statements and leave most decisions on log placement to developers should be enhanced. For example, to provide recommendations around log-prone code snippets, to provide message prompt if log statements are missing from critical log-prone code snippets, etc. Nevertheless, the interviewees also expressed their doubts about such "powerful" logging tools–perhaps just too complex to be realized.

Guidelines. In fact, guidelines regarding logging practice triggered heat discussion. As Table IX shows, on one hand, since none of our interviewees have guidelines currently, most interviewees anchor their hope on effective guidelines. On the other, some interviewees also expressed their concerns about the effectiveness of guidelines. They think such guidelines may be unnecessary and useless.

TABLE IX
INTERVIEWEES' ATTITUDES ON LOGGING GUIDELINES

Interviewee	Attitude
HD1	Positive. It is good to have general guidelines.
HD2	Positive. Guidelines are necessary for newcomers.
IC1	Negative. Most guidelines can not be applied in current projects.
IC2	Negative. Guidelines are hard to define and usually useless.
SG1	Positive. Guidelines help to avoid injection of log related issues.
SG2	Positive. Guidelines are necessary for newcomers.

Finding 4: Developers resort to enhanced supporting tools and logging guidelines to improve current logging practice. However, they have divergent viewpoints towards the effectiveness and feasibility of these methods.

V. DISCUSSION

In this section, we discuss the implication of our findings and possible solutions as well.

A. The Importance of Logging Practice and Developers' Perception

As an important part of operation data produced by logging techniques, logs are playing a more and more important role in modern software development. For example, as a new software development and operations paradigm, DevOps [3], [25] and AIOps [26] are attracting more and more attention from practitioners. To monitor either a large software system or many microservices, high-quality logs are critical which requires high-quality logging practice. This study reveals one fact that logging is pervasively used in software development and maintenance in *XH*. Nearly all the developers know logging practice and have more or less experience in logging practice. Besides, different logging tools as one of the daily development facilities have been widely adopted.

Developers are aware of the importance and value of logging practice, as one developer mentioned that "Logs, in most cases, are the only available data for us to diagnose failures, so we have to insert log statements in our code." This statement is consistent with some opinions from other researchers that logging is one of the few mechanisms for gaining visibility of the behavior of the software system [2], [7], [8], [27]. Zhu et al. [8] also pointed out that the pervasive existence and active modifications of log statements reveal that logging plays an indispensable role in software development and maintenance.

B. Inconsistencies between Developers' I&C and Real Log Placement

The results of our study imply that developers' *I&C* are often not properly reflected in the source code, i.e., there are inconsistencies between the original *I&C* and the real implementation in source code. Besides, source code may be modified for many reasons, which further increases the opportunities for the aforementioned inconsistencies. Developers are facing several challenges when carrying out logging practice,

among which *"where to log?"* and *"what to log?"* are still two major unsolved challenges, not to mention a proper reflection of developers' *I&C*.

Several studies attempted to address these challenges. For example, He et al. [28] focused on the natural language descriptions of log statements for the sake of filling in the gap of *"what to log?"*. Zhao et al. [2] proposed a tool that can automatically place log statements into source code. Li et al. [12] proposed an approach to help developers determine the appropriate log level, and so on. Apparently, these "bottom-up" efforts may not be able to totally eliminate the inconsistencies between developers' *I&C* and log statements in source code, since there lacks a comprehensive and systematic perspective towards log placement. Researchers also tried a "top-down" strategy to improve the logging practice. For example, Lal et al. [29] suggested that log statements have a trade-off between cost and benefit and it is important to optimize the number of log statements in the source code. Nevertheless, this strategy also faces challenges, as pointed out by Fu et al. [4] that optimizing log statements in the source code is a non-trivial task, and software developers often face difficulty in it.

C. Guidelines for General-Purpose Logging

Logging guidelines seem to be a useful solution. With recognizing the importance of guidelines for logging practice, a number of studies attempted to propose the best practices for logging. Cinque et al. [30] proposed a set of rules to enrich traditional logging for the sake of improving the quality of logged failures and easing the coalescence of redundant or equivalent data. Fu et al. [4] categorized logged code snippets and summarized factors need to be considered for logging decisions. Some studies also propose guidelines for logging practice as future work. Cinque et al. [31] planned to encompass the definition of a wider set of guidelines to improve the suitability of logs for the analysis of software faults. There are also several blogs discussing the best or worst logging practices, e.g., [32], [33]. Some world-leading software companies have also introduced internal guidelines for logging practice, e.g., Alibaba [22]. Most of them focused on logging practice for general purpose, i.e., logging without any specific *I&C* under a specific context.

However, in our opinion, effective guidelines for general-purpose logging might not exist. From the perspective of software developers, logging is an approach that providing informative data for subsequent diagnosis or analysis. These analysis objectives can be divided into several different categories, e.g., performance diagnosis, system profiling, etc. Different analysis techniques and purposes require different types of information. In this sense, the goal to design general-purpose logging guidelines might not be practical and feasible, which to a certain degree explains the conflicting attitudes on the usefulness of logging guidelines among interviewees. Moreover, as Finding 3 implies, the situation of exceptional cases obviously increases the difficulty (if not impossible) to establish a suitable guideline.

As revealed in our study and reflected in internal coding guidelines such as [22], it seems that "general-purpose" guidelines can only address simple issues such as naming, on-off switch for log level, etc. which may only help newcomers.

D. Shift-Left of Logging Practice

Current research primarily focuses on the improvement of existing log statements from mainly two aspects, *where to log?* and *what to log?*. However, coping with source code to address business logic easily attracts developers' attention and impede developers from forming systematic perspectives towards log placement. In this sense, logging practice should be considered to be shifted left to upstream development phases such as design and requirement where developers may have a better position to think about the overall log placement systematically.

In fact, some researchers already have noticed this topic. For example, Cinque et al. [30] proposed a set of rules to enrich traditional logging. Similarly, the authors advocated that the proposed rules should be followed at design time. In short, the implementation of logging should be designed along with the system design.

VI. Threat to Validity

In this section, we present and discuss the validity threats for this study.

A. Construct Validity

Construct validity reflects the degree to which the measures and evidence in the study accurately represent the research intention. The following issues associated with construct validity have been identified:

a) Only examining the unlogged code snippet for consistency checking: When comparing the *I&C* and the source code, we only involved the "absent logging" inconsistency, i.e., an unlogged code snippet which should have log statement is determined as inconsistency. However, the opposite situation, i.e., a logged code snippet containing unnecessary log statements is not determined as an inconsistency. Nevertheless, this setting only decreases the number of inconsistencies, which may not impact our conclusion given the fact that plenty of inconsistent cases of "absent logging" have been identified.

b) Only verifying the occurrence of log statements: We only verify whether a log statement exists in the code snippet as the corresponding developer described without examining the content of the log statement. Apparently, the *I&C* of the developer means far more than the occurrence of a log statement in a specific snippet. By involving this factor will inevitably increase the number of inconsistencies between *I&C* and log placement. In this sense, the findings still hold true.

c) Only involving the unlogged code snippets near logged ones: The tool and commands used to locate code snippets for further manual checks are based on the precondition that there should exist at least one log statement in the code snippet. In this sense, those code snippets without any log statement (no matter there should be or not) are excluded in

our study. Similarly, this choice will only decrease the number of inconsistencies and will not impact our conclusion in this paper also.

B. Internal Validity

Internal validity is the extent to which a study establishes a trustworthy cause-and-effect relationship between a treatment and an outcome. The following issues associated with internal validity have been identified:

a) The relatively small number of interviewees: Another threat to the validity is that we only involved 6 interviewees from three projects in our study. Therefore, results and findings in this study may not reflect all the real status regarding logging practice in *XH*. However, we involved one-third of the core projects in *XH* in our study. Besides, the interviewees all have multiple years of working experience in this company, including all the team leaders in these three projects. This setting provides a credible source of information. Meanwhile, we focus on the difference between various evidence sources in this study. In this sense, this threat could be largely mitigated.

C. External Validity

External validity reflects the degree to which we can generalize the results to other contexts. The following issues associated with external validity have been identified:

a) Java as the sole programming language: One possible threat to the external validity might be that we only involved the projects developed with Java language. Developers' *I&C* towards logging practice may differ when using other programming languages. For example, different programming languages may have different mechanisms to manage memory, which may require different ways to write and put log statements for performance diagnosis. Nevertheless, Java is one of the most adopted programming languages in many companies. In this sense, our results towards developers' I&C in industry can be valuable.

b) Big-data company as the only business type: Another possible external validity can be that we only observed developers in one big data company. Developers in different companies in other business domains may have different *I&C* towards logging practice and thus the inconsistencies between the *I&C* and the log statements may also differ to a certain degree. In this sense, it is important to carry out similar studies with different types of companies so that we can deepen our understanding of logging practice.

VII. Conclusion

With the continuous increasing scale and complexity of software systems, understanding the dynamic behavior of software systems via logs is critical in many software practices such as debugging, performance-optimizing, etc. Logs are footprints of the running software systems, which are produced by the log statements written by developers. Apparently, the log statements are supposed to reflect developers' *I&C* to put them in various places in software source code. However, although there are plenty of studies on logging practice, very little research work focused on the developers' *I&C* behind each log statement.

We carried out a case study that includes a series of interviews and corresponding code analysis at *XH* company to understand the developers' *I&C* and further identify possible inconsistencies between the *I&C* and real log statements. As one single case study, we do not intend to generalize findings. Nevertheless, some interesting facts and implications based on our findings can be summarized as the following:

First, as a regular and daily activity in software development, logging practice gained widespread awareness among developers. Tools and guidelines are used to address simple issues regarding logging practice, which may not help sophisticated developers.

Second, by focusing on developers' *I&C* behind each log statement, we provided one possible explanation for the non-ideal adoption status. Due to various reasons, for example, lacking supporting facilities or the version evolution of source code, the developers' *I&C* are usually poorly reflected in the source code, rendering questionable capability to capture intended information about system behaviors via logs.

Last but not least, the divergent attitudes on the effectiveness and feasibility of logging tools and guidelines imply that to reflect the *I&C* of developers in log placement might not be an easy problem to solve. As a matter of fact, tools and guidelines are not new ideas, but to the best of our knowledge, they also have very limited effects to improve the logging practice in industry. In this sense, research attention may be needed to explore new mechanisms and approaches to represent *I&C* and guide, verify and monitor their implementation in log placement. At this preliminary stage, we suggest several promising research directions as follows:

(1) A large-scale empirical study that focuses on developers' *I&C* towards logging practice in different companies across different business domains, which may help us establish a more complete and comprehensive understanding of the adoption status of logging practice as well as the developers' *I&C* in industry.
(2) Taking the content of log statements into account in the mutual comparison of *I&C* and log statements to better identify the inconsistencies.
(3) To explore approaches to include, reflect and manage the developers' *I&C* in the source code.

Acknowledgment

This work is partially supported by the National Key Research and Development Program of China (No. 2019YFE0105500).

References

[1] 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 '12)*. IEEE Press, Jun. 2012, pp. 102–112.

[2] 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 '17)*. ACM, Oct. 2017, pp. 565–581.

[3] G. Rong, S. Gu, H. Zhang, D. Shao, and WanggenLiu, "How is logging practice implemented in open source software projects? a preliminary exploration," in *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC '18)*. IEEE, Nov. 2018, pp. 171–180.

[4] 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 (ICSE Companion '14)*. ACM, Jun. 2014, pp. 24–33.

[5] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry practices and event logging: Assessment of a critical software development process," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2 (ICSE '15)*. IEEE Press, May 2015, pp. 169–178.

[6] W. Shang, M. Nagappan, A. E. Hassan, and Z. M. Jiang, "Understanding log lines using development knowledge," in *Proceedings of International Conference on Software Maintenance and Evolution (ICSME '14)*. IEEE, Sep. 2014, pp. 21–30.

[7] R. Ding, H. Zhou, J.-G. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie, "Log2: A cost-aware logging mechanism for performance diagnosis," in *USENIX Annual Technical Conference*. USENIX Association, Jul. 2015, pp. 139–150.

[8] 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-Volume 1 (ICSE '15)*. IEEE Press, May 2015, pp. 415–425.

[9] D. Yuan, S. Park, P. Huang, Y. Liu, M. M.-J. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: Enhancing failure diagnosis with proactive logging," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, Oct. 2012, pp. 293–306.

[10] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou, "The game of twenty questions: Do you know where to log?" in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, May 2017, pp. 125–131.

[11] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 1, p. 4, Feb. 2012.

[12] H. Li, W. Shang, and A. E. Hassan, "Which log level should developers choose for a new logging statement?" *Empirical Software Engineering*, vol. 22, no. 4, pp. 1684–1716, Aug. 2017.

[13] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, Feb. 2018.

[14] H. Li, W. Shang, Y. Zou, and A. E. Hassan, "Towards just-in-time suggestions for log changes," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1831–1865, Aug. 2017.

[15] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," *Journal of Software: Evolution and Process*, vol. 26, no. 1, pp. 3–26, 2014.

[16] B. Chen and Z. M. J. Jiang, "Characterizing and detecting anti-patterns in the logging code," in *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, May 2017, pp. 71–81.

[17] Z. Jia, S. Li, X. Liu, X. Liao, and Y. Liu, "SMARTLOG: Place error log statement by deep understanding of log intention," in *Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering (SANER '18)*. IEEE, Apr. 2018, pp. 61–71.

[18] H. Anu, J. Chen, W. Shi, J. Hou, B. Liang, and B. Qin, "An approach to recommendation of verbosity log levels based on logging intention," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 125–134.

[19] Z. Li, T.-H. Chen, J. Yang, and W. Shang, "Dlfinder: Characterizing and detecting duplicate logging code smells," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 152–163.

[20] F. Baccanico, G. Carrozza, M. Cinque, D. Cotroneo, A. Pecchia, and A. Savignano, "Event logging in an industrial development process: Practices and reengineering challenges," in *Proceedings of 2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW '14)*. IEEE, Nov. 2014, pp. 10–13.

[21] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, and M. Yang, "Chukwa: A large-scale monitoring system," in *Proceedings of CCA*, vol. 8, 2008, pp. 1–5.

[22] Alibaba, "Alibaba Java coding guidelines," https://alibaba.github.io/Alibaba-Java-Coding-Guidelines/, accessed: 2019-12-08.

[23] V. R. Basili, G. Caldiera, and H. D. Rombach, "Goal, question metric paradigm. encyclopedia of software engineering, vol. 1," 1994.

[24] 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, 2012.

[25] H. Akshaya, J. Vidya, and K. Veena, "A basic introduction to devops tools," *International Journal of Computer Science & Information Technologies*, vol. 6, no. 3, pp. 05–06, 2015.

[26] Broadcom, "The definitive guide to aiops," https://docs.broadcom.com/docs/the-definitive-guide-to-aiops, accessed: 2019-12-08.

[27] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[28] 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 (ASE '18)*. ACM, Sep. 2018, pp. 178–189.

[29] S. Lal, N. Sardana, and A. Sureka, "Analysis and prediction of log statement in open source Java projects," *Buenos Aires, Argentina*, p. 65, May 2017.

[30] M. Cinque, D. Cotroneo, and A. Pecchia, "A logging approach for effective dependability evaluation of complex systems," in *Proceedings of the Second International Conference on Dependability (DEPEND '09)*. IEEE, Jun. 2009, pp. 105–110.

[31] M. Cinque, D. Cotroneo, R. Natella, and A. Pecchia, "Assessing and improving the effectiveness of logs for the analysis of software faults," in *Proceedings of 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN '10)*. IEEE, Jun. 2010, pp. 457–466.

[32] L. Tal, "9 logging best practices based on hands-on experience," https://www.loomsystems.com/blog/single-post/2017/01/26/9-logging-best-practices-based-on-hands-on-experience, Jan. 2017, accessed: 2019-12-08.

[33] J. Skowronski, "30 best practices for logging at scale," https://www.loggly.com/blog/30-best-practices-logging-scale/, Jan. 2017, accessed: 2019-12-08.