Event Logging in an Industrial Development Process: Practices and Reengineering Challenges

Fabio Baccanico*, Gabriella Carrozza‡, Marcello Cinque*†, Domenico Cotroneo*†,
Antonio Pecchia* and Agostino Savignano*

*Critiware S.r.l. – Via Carlo Poerio 89/A, 80121 Naples, Italy
†Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione
Universitá degli Studi di Napoli Federico II – Via Claudio 21, 80125, Naples, Italy
‡ Selex ES S.p.A - A Finmeccanica Company – Piazza Monte Grappa 4, 00195 Rome, Italy
Contact Email: antonio.pecchia@critiware.com, gabriella.carrozza@selex-es.com

***Abstract*—This paper discusses our preliminary analysis of event logging practices adopted in a large-scale industrial development process at Selex ES, a top-leading Finmeccanica company in electronic and information technologies for defense systems, aerospace, and land security. The analysis aims to support log reengineering activities that are currently conducted at SELEX ES. At time being, some of the issues described in the paper have been fixed by system developers. Analysis encompasses total around 50+ millions lines of log produced by an Air Traffic Control (ATC) system. Analysis reveals that event logging is not strictly regulated by company-wide practices, which results into heterogeneous logs across different development teams. We introduce our ongoing effort at developing an automatic support to browse collected logs along with a uniform logging policy supplementing the reengineering process.**

***Keywords*-Event logging, Development Process, Air Traffic Control, Logging Practices.**

I. Introduction

Event logging is a well-established practice to collect information about the behavior of a computer system in a set of logs. Logs are textual files containing the records of events occurring during the execution of a given system: information provided by **event logs** is widely used for several system management tasks. Over the past decades both academic and industrial organizations have widely recognized that event logging is valuable for a variety of purposes and dependability-related tasks, such as root cause analysis, error and failure characterization, anomaly detection, error debugging, performance diagnosis, analysis of security alerts [1], [2], [3], [4], [5], [6], [7]. In spite of the importance of the tasks based on logs, several studies observed that event logging lacks in terms of systematic design and implementation practices.

Key decisions about **log production** and **management** are usually left to the late stages of the system life cycle (e.g., coding). Implementation of the logging mechanism rely on the experience of developers and programmers. As a result, information provided by collected logs might be subjective [8] and highly unstructured [9]. At the same time, the lack of standardized logging solutions across vendors is a crucial concern in case of large-scale development processes. Different software items, such as operating system, middleware, and applications, log with different formats and without any form of cooperation. Issues with real-world industrial logs are due to limited information and practices indicating *how* and *what* to log [10], [11]. Systematic logging practices play a crucial role because the correct placement of the logging instructions can increase accuracy of logs at runtime [12], [13].

This paper discusses our preliminary analysis of event logging practices adopted in a **large-scale industrial development process** in the Air Traffic Control (ATC) domain, which involves a community of programmers, system integrators and maintenance personnel accounting for total around 100 units. The ATC system is developed by Selex ES[1]. Logs considered in this study play a critical role across the development process because they represent a key information source to investigate failures occurring during production [14]. We analyze total **50+ millions** lines of log generated by around 60 Computer Software Configuration Items (CSCI)s, which implement the system functions. Analysis indicates that event logging is not regulated by company-wide practices, which results into heterogeneous logs across different development teams. A closer look into the data revealed that event logging serves three major **objectives**, i.e., *event reporting*, *application state dump* and *execution tracing*. The paper introduces our ongoing effort at developing an automatic tool to browse collected logs along with a uniform logging policy aiming to support log reengineering activities that are currently conducted at SELEX ES for systematizing logging practices.

II. Assessment

A. Logging and Analysis Practices

The reference ATC system integrates a wide range of products and tools to comply with heterogeneous operational requirements and Air Traffic Management environments. The system consists of around 60 CSCIs, which implement three major product families, i.e., (i) *Business Logic* (BL), (ii) *Middleware* (MW), and (iii) *Human-Machine Interface* (HMI). CSCI products are developed, integrated and maintained by different teams. The software development process is partially

[1] http://www.selex-es.com/

978-1-4799-7377-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ISSREW.2014.69

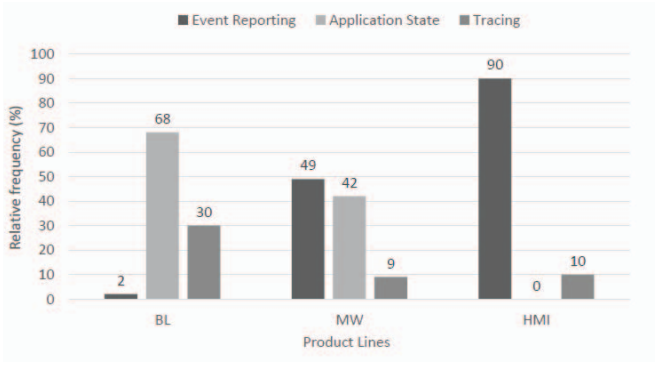

Fig. 1. Relative frequency of log patters, by purpose, for each product line.

regulated by company-wide documented practices; moreover, each product family has additional practices coping with needs and objectives pursued by a given development team.

Based on interviews with different teams, it emerged that developers within a given product family share few rules regarding message structure and logging architecture. This is a common drawback in a large scale development process. Moreover, it should be observed that the logging mechanism of each product line has not been developed with the aim of sharing logs across different teams in the same organization. As a result, the three main product lines are characterized by different log structures and different log production mechanisms, which is a known issue for any large scale development process. SELEX ES is currently devoting a strong effort to standardize logging practices in the reference scenario. **Key challenges** involve the following aspects:

- enforcement of common logging rules by means of a **company-wide policy**: rules are currently demanded to few developers;
- evolution of logging practices that have been influenced by both individual and common needs;
- documentation of current logging mechanisms and information sharing among different product lines.

Mentioned issues strongly impact effectiveness of log analysis. Interaction with system integrators at SELEX ES allowed to gain insights into analysis steps that are conducted to diagnose failures affecting ATC components. Typical analysis steps include (i) retrieving the text file(s) of failed modules containing the log produced at time the failure occurred, (ii) using command-line and terminal programs (e.g., `grep`, `vim`) to extract patterns of interest, (iii) creating a copy of useful log lines in a text file and keep them for further analysis, (iv) **correlating** information coming from logs of different components, in order to establish the fault location. The latest step exacerbates the need for navigating across data generated by different CSCIs, which turned out being a time-consuming activity due to the heterogeneity of collected logs. Ongoing reengineering activities aim to overcome mentioned issues by systematizing production strategies and format issues of logs available in the reference domain.

B. Data Analysis

The data analysis has been conducted in two phases: the **pattern extraction** phase and the **classification** phase. The goal of the first phase is to build a model for the log files, by means of *clustering* of log messages. Raw logs provides big amounts of data, much of which is repeated or redundant. Before using these logs for analysis purposes, it is useful isolating clusters of messages (i.e., patterns). Logs are usually composed by variable and constant fields. Variable fields are the words that identify manipulated objects or states of the program, for example IP and MAC addresses, threads IDs, dates, user names. Constant fields are the words that do not change across the same pattern and contains information to describe the message type. The second phase is characterized by the extraction of significant features from patterns. Each pattern has been manually inspected, tagged according to its purpose, and finally classified on the basis of its semantic and structure.

Logs are mainly adopted to support debug and system integration activities: integrators and developers use collected logs to reduce the amount of code to be inspected in case of failures observed during operations. Logs serve three different goals within the reference ATC scenario: (i) **event reporting** (e.g., information, error, warning messages), (ii) **application state dump**, (iii) **application tracing**. It is worth noting that logs from different product families focus on different aspects. Logs have been evolving into different directions, since each product line had different needs. As depicted in Fig. 1, logs from HMI are mainly used for event reporting, while this purpose is nearly absent in logs from BL and MW. Application state dump is the predominant purpose in logs from BL, while it has the same frequency of application tracing in logs from MW.

III. Format and Semantic Issues

We analyzed and compared each pattern obtained in the data analysis phase with the aim of gaining insights into format and semantic issues of available log files. Table I shows several patterns by product line, i.e., BL, MW, HMI. Each line is identified by a numeric ID reported in the leftmost column of Table I. We observed that collected logs are strongly heterogeneous either in terms of formats and semantics, probably due to the lack of agreement across different development teams. Relevant issues of available logs are discussed in the following. Some of the mentioned issues have been fixed by system developers.

Lack of unique time format. Each product line has its own time representation. For example, the BL family uses a timestamp in a form of `04 05:23:05.791`, while the MW uses `2014-04-03 17:38:19.486` and the HMI use `03/06/2014 20:26:34`. The adoption of different time formats, makes it hard to browse logs belonging to different families. It is worth noting that the time representation varies even within the same line in the log, as it can be observed by analyzing lines 11 and 18 of Table I.

TABLE I
EXAMPLES OF LOG PATTERNS.

ID	Family	Timestamp	Sample Message
1	MW	2014-04-03 18:17:55.010	POS TIMECOUNT SECTOR ACTION FLI CALLSIGN
2	MW		0 1 0 3 352 *
3	MW		1 1 0 3 815 *
4	MW		2 1 0 3 503 *
5	MW		3 1 0 3 332 *
6	HMI	2014-04-03 17:38:19.486	–CVawConflict– CVawConflict::resetConflictVawTable call
7	BL	04 12:15:32.669	[CSCI1-03-011] ** ** Z ACU C 05550 Apr 04-11:32:36 IG 0001 IN:00 OUT:00 1 [0.0] R:1
8	HMI	03/06/2014 20:26:35	- ATC_SUP Warning: SUP_IMsend: Send message to HDI (applic_id = 0 - order_id = 162 - size = 62)
9	BL	04 12:16:37.849	[CSCI1-00-017] CSCI: FillItemVolo: Sezione COP - OUT: " - Type: 0 - Range: -5232 - Bearing: -5230
10	BL	04 12:16:37.849	[CSCI1-00-017] csci: fillFdpSection: Id: ** - KeyId: 5760646
11	BL	15 11:19:00.383	[CSCI2–034] Event: CheckExpired: currTime Apr 15 2014 11:19:00 - Range [Apr 15 2014 11:19:00 / Apr 15 2014 11:19:00]
12	BL	15 11:19:00.383	[CSCI2-034] Event: CheckExpired: current Time lesser than Range
13	BL	15 11:19:00.393	[CSCI2-034] CCheckFplChanges: LoadFpl: FLIGHTID: *
14	HMI	07/15/2013 11:43:04	- ATC_FLIGHT Message: operator reset hook of flight *
15	BL	15 11:19:00.485	[CSCI2-034] CSCI - FlightId: * - ItemVoloPoint: - PointType:
16	BL	17 16:29:58.969	[CSCI3-000] NEW CONFLICT: viol_ID 17534 - traj1_ID 2977 - seg1_ID 13946 - Last Point x=[-56.000000] y=[-88.000000]
17	BL	17 14:31:10.530	[CSCI3-000] CSCI_addSegmentViolation: violID 16684, tStart: 15:09:06 , traj1ID 2795:0 seg1ID 55260, Last Point (-10825,-16144)
18	BL	04 12:15:32.708	[CSCI4-07-016] Order: HandleTrajectoryPrediction: FPLID THY** - TerTime: Apr 04 12:32:54
19	BL	04 12:15:32.718	[CSCI4-016] COrder: SetStateRelatedTrj [THY**] - ROUTEREPORTFIX [CLK**] After SetStateRelateOrder - State[7]
20	BL	17 16:22:55.021	[CSCI5-000] FindRisk: FL1=(THY**) SectIDSUCC=(***) - CFL1=330.000000 - XFL1=330.000000

Different syntax and representation is often used to represent the same concept. Two instances of this issue are discussed in the following by means of the examples reported in Table I. We observed that the same component is identified either in uppercase or lowercase (adoption of either `CSCI` or `csci`, such lines 9 and 10); similarly, a different syntax (i.e., `currTime`, in log 11 and `Current Time`, in log 12) is adopted to represent the same event. The adoption of different identifiers requires a deep knowledge of concepts and representations to navigate the log.

There is no a common way to represent value-key pairs. A closer look into available data revealed the adoption of a variety of formats. For example, Table I shows examples where logs are written as `value=key` (line 8), in other as `value:key` (line 9) or `value [key]` (line 19). We found out around 8 different ways to represent values. More importantly, different representation are often used in the context of the same line (e.g., line 20). The representation is not uniform across collected logs. For example, lines 16 and 17 use a different data-structure to represent the same value. The lack of a standardized practices makes it hard to automatically extract values from the log file.

The same information is sometime sparse across many lines. For example, lines from 2 to 5 reports a tabular representation of the fields reported by line 1 of Table I. This practice is adopted for the sake of better visualization of the information in the log; however, it is hard to infer the lines related to the same information. In this case, the adoption of a sequence number would have strongly helped at discriminating lines providing the same information.

The adoption of a severity value for each message in the log is not a common practice across different product lines. As shown in I, CSCI belonging to BL and MW families do not adopt a severity indication. We observed that log messages produced by HMI products produce a severity value; however, there are no strict policies regulating the adoption of given severity values, which are established by individual developers. Severity is a known feature to support message filtering.

The majority of these problems impact both manual and automatic log analysis. Manual log analysis is complex since often the team that analyzes the contents of the log is not the same team that coded the log statements. For example, during the analysis of failures affecting different components of the system, (i) syntax and semantic heterogeneity of collected logs, and (ii) limited information sharing among developers belonging to different product lines, make troubleshooting and log forensics complex and time-consuming. Further problems arise with automatic analysis, due to the presence of unstructured and domain-specific information that impact the design of a comprehensive log analysis tool. For instance, different timestamp format should be converted to a common representation before analysis. Results inferred by means of logs are potentially hard to be interpreted due to concepts represented with different syntax (different name, or different key-value syntax); this operation would require a definition of a thesaurus to group all the different terms that represent the same concept.

IV. ONGOING WORK AND CHALLENGES

Our efforts at supporting log reengineering tasks conducted at SELEX ES are summarized in the following. We identified two main challenges, i.e., the definition of a *logging policy* and the development of a *logging API*. The **logging policy** encompasses the definition of a set of rules shared across all the product families. Rules encompass several aspects, such as placement of the logging code (i.e., *where* and *what* to log), format, and semantics of the information recorded in the log file. The definition of a logging policy is not a trivial task because logs from different families serve a variety of purposes, as explained in section II-B. For instance, a log message that is used for error reporting may have a completely different structure than a log message used to trace the application state or the control-flow of the program. Additional difficulties arise when defining *where* to log. An error-reporting message may be produced after checking the validity of an assertion, while the best way to dump a portion

of the application state may be by producing messages within the loop that is modifying such state. At the same time, the policy is expected to be exhaustive, since it has to cover all the possible use cases of event logs.

While the development of a policy aims to systematize common logging practices across different teams, a **logging API** would help enforcing the policy. It should be observed that the adoption of a logging API requires a strong reengineering effort because any current log statement has to be replaced with its equivalent API calling. Adding, removing or updating a logging point involves a change of the source code: the cost of modifying a logging point is around 1,000$, considering the effort for regression testing. A reengineering strategy must cope with the actual return of investment (ROI) that can be achieved by means of better event logging strategies.

We are also developing an analysis tool to speed-up forensics of collected data. The tool aims to support the navigation of logs collected across different product families. The tool provides prototype mechanisms supporting **log analysis and visualization**. These are the most important steps of log analysis conducted by the system administrator and require a manual effort. The objective of our tool is to provide assistance in (i) quick data visualization in order to give mechanisms to surf data quickly, (ii) definition of simple and complex filters in order to allow the system administrator to focus the attention where it need, and (iii) correlation of information across different components in order to verify and analyze the propagation of the information. Moreover, the tool can help defining rules supporting pattern matching, comparing numerical thresholds, or regular expressions, matching a large number of log entries. The final goal of the tool is to provide mechanisms to support real-time monitoring of system logs through analysis of features and statistics defined by operator using a formal language.

V. Conclusion

Almost all system operators, integrators, and developers agree that event logs are a primary source of information for diagnosing problems. However, preliminary lessons learnt in the context of our activity indicate that the logging process is strongly dependent from individual developers. The lack of a systematic logging approach has become a serious drawback over the years and poses a major reengineering challenge at SELEX ES. Given the importance of event logging in the reference scenario, our future work will face the definition of effective policies aiming to systematize the implementation of the logging mechanism across different product families.

References

[1] Dong Tang and R.K. Iyer. Analysis of the vax/vms error logs in multicomputer environments-a case study of software dependability. In *Software Reliability Engineering, 1992. Proceedings., Third International Symposium on*, pages 216–226, Oct 1992.

[2] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 426–435, New York, NY, USA, 2003. ACM.

[3] C. Di Martino, F. Baccanico, J. Fullop, W. Kramer, Z. Kalbarczyk, and R. Iyer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Proc. of 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), 2014*, 2014.

[4] Santonu Sarkar, Rajeshwari Ganesan, Marcello Cinque, Flavio Frattini, Stefano Russo, and Agostino Savignano. Mining invariants from saas application logs (practical experience report). In *Dependable Computing Conference (EDCC), 2014 Tenth European*, pages 50–57, May 2014.

[5] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Geetika Goel, Santonu Sarkar, and Rajeshwari Ganesan. Characterization of operational failures from a business data processing saas platform. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 195–204, New York, NY, USA, 2014. ACM.

[6] A. Pecchia, A. Sharma, Z. Kalbarczyk, D. Cotroneo, and R. K. Iyer. Identifying compromised users in shared computing infrastructures: A data-driven bayesian network approach. In *Proceedings of the Int'l Symposium on Reliable Distributed Systems (SRDS)*, pages 127–136. IEEE Computer Society, 2011.

[7] Antonio Pecchia, Domenico Cotroneo, Rajeshwari Ganesan, and Santonu Sarkar. Filtering security alerts for the analysis of a production saas cloud. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, Washington, DC, USA, 2014. IEEE Computer Society.

[8] M. Kalyanakrishnam, Z. Kalbarczyk, and R. K. Iyer. Failure data analysis of a LAN of windows NT based computers. In *Proceedings of the Eighteenth Symposium on Reliable Distributed Systems (18th SRDS'99)*, pages 178–187, Lausanne, Switzerland, October 1999. IEEE Computer Society.

[9] C. Lim, N. Singh, and S. Yajnik. A log mining approach to failure analysis of enterprise telephony systems. In *International Conference on Dependable Systems and Networks (DSN 2008)*, Anchorage, Alaska, June 2008.

[10] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, 39(6):806–821, 2013.

[11] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. International Conference on Software Engineering, June 2014.

[12] M. Cinque, D. Cotroneo, and A. Pecchia. A logging approach for effective dependability evaluation of complex systems. In *Proceedings of the 2009 Second International Conference on Dependability*, DEPEND '09, pages 105–110, Washington, DC, USA, 2009. IEEE Computer Society.

[13] A Pecchia and S. Russo. Detection of software failures through event logs: An experimental study. In *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, pages 31–40, Nov 2012.

[14] M. Cinque, D. Cotroneo, R. Della Corte, and A. Pecchia. Assessing direct monitoring techniques to analyze failures of critical industrial systems. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, Nov 2014.