Rule Based Event Log Generation Technique using Reverse Engineering

K.Shanmugapriya, C.Geetha,Mary Linda I, D.Vimala

Abstract*: *In many software systems logging has been implemented inaccurately, their effectiveness during the maintenance period to identify the failures and address them quickly is very less. This in turn increases the software maintenance cost and reduces reliability of the system as many errors are unreported. This paper aims at proposing and studying a rule based approach to make the logs more effective. The source code of the target systems gets reverse engineered and acts as the primary input for this approach to introduce the automated logs into the source code. This is instrumented by a logger code driven by a set of predefined rules which are woven around the life cycle of the system entities. The validity of the approach is verified by means of a preliminary fault injection experiment into a real world system.

.

Keywords*: *fault detection, logging, fault diagnosis.

I. INTRODUCTION

Failures are inevitable in software systems and are costing companies and consumers large amounts of money.[5] System and application logs are the primary mechanism of debugging when some failure occurs. It is important that the log reporting has to be accurate in order to identify the cause of failure by the production support engineers. Though many systems in real time tend to have robust logging, failure detection is still a challenge especially in large systems which nowadays invariably exists in complex environment interacting with multiple other systems. Unanticipated data conditions and environmental settings may cause logical errors and other failures to appear for the first time when they are into production.

Failure detection issues are not only common for just released systems but also for decade long systems which are running successfully where small patches towards enhancements could cause difficult to trace issues. Such problems often require quick turn around and accurate fixes which is not possible if the logs are inaccurate. Whatever be the reason for the failure, inaccurate logging is necessarily a problem that needs to be holistically addressed, the solution is multidimensional in nature involving log formats, what to log, where to log and how. This research work is providing the approach on the solution on where and how to parts and does not focus on the log formats. The approach partially helps proactive identification of certain issues in situations where no exception is thrown. Abnormal execution in real time systems tend to break the flow of control and escape the error handling code many a times.

The logging rules have been tailored and tested against an object oriented system and their validity has been assessed by a preliminary manual testing. Reverse engineering or the system expertise knowledge has to be used for such already running systems to identify the critical system entities.

Section II provides a brief on literature review conducted as part of this study. The proposed solution is discussed as part of section II and Section III. Section IV describes the study carried out and the results and analyzed in Section V.

II. BACKGROUND WORK

[1]. Detailed study has been carried out on various techniques by us as part of the study work [2] but discussed a few briefly here. There are various log based solutions but most of them compliments each other for addressing different sub areas. As part of [7] solution, each point of logging is identified and the control flow and data flow is tracked till that point within the function. The variables involved in the condition and each memory location involved are the information suggested to be added to the log point.

[2]. SVM technique is used [9] to predict faults from log files. Random Indexing is used to represent sequences of operations extracted from log files, Support vector machine is used classify them as either failure or success.

[3]. Spectrum based fault localization technique [10] is suggested which can be applied on a program without much knowledge about the program hence suits best for the testing schemes.

[4]. The rules based approach[1] suggests automated log insertion in a structured way. The important locations for logging are function start and end, start and end of an entity life cycle, interactionstart and end. Any abnormality on this flow to be automatically detected and alerted. This workremains as the main motivation for the current study, similar approach enhanced and studied in different technology context along with reverse engineering

Revised Manuscript Received on July 22, 2019.

K.Shanmugapriya, Department of Computer Science and Engineering, Bharath Institute of Higher education and research, Chennai, India
C.Geetha, Department of Computer Science and Engineering, Bharath Institute of Higher education and research, Chennai, India
I.Mary Linda, Department of Computer Science and Engineering, Bharath Institute of Higher education and research, Chennai, India
C.Geetha, Department of Computer Science and Engineering, Bharath Institute of Higher education and research, Chennai, India

.

III. PROPOSED SOLUTION

Log enhancements are carried out in two steps, initial step of identifying the components and their call tree to implement the functionality (reverse engineering) and the subsequent step being the code enhancement based on the predefined set of rules. The aim of the reverse engineering process is mainly to understand the entities present in the system and how they are related to each. The entity information is subsequently used by the log generation component. The proposed logging approach leverages a separate logger framework which is decoupled from the system code and is designed to get executed asynchronously to ensure the system performance is intact. The framework proactively checks for potential hangs and infinite loops. A simple architecture representation is depicted in Figure 1. Code generation environment is used to revere engineer and insert the logging code into the files. The application binary has to be kept in the file repository and fed to the reengineering component.

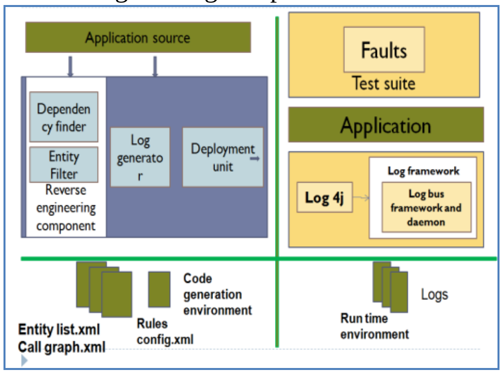


Figure 1: Solution Architecture

Log generation takes the entity and the call graph information as the input. For each entity log generation component checks the applicable rules and transforms the code to have the log code inserted. Logging is done using Logbus-ng framework[4]. It consists of a daemon and a bus service. Once an event is received the library adds information such as process id, entity and writes to a shared memory. The rules are defined in such a way that each entry and exit points of each method and entity lifecycle and most importantly in every block where there may be a potential error.

IV. CODE INSTRUMENTATION

The Rule based logging framework consists of two major modules. The reverse engineering component does the job of deriving entity list and call graph from the system binary under study. For each entity found in the entity list, log generator inserts logs as driven by the rules configuration. Log framework is an external component library used as part of this project as is without any modification.

A simple dependency finder tool has been used in this project to identify the list of entities and their call graph[3]. Appropriate filter has been developed that gets only the components needed for the needed scope by mentioning the package and class patterns. A set of rules are defined as in table 1. The explanation of what each rule means and how log bus behaves for each can be found from [1]. Each rule at this table is extended to define templates to take care of various semantics. The rule EXC inserted try catch blocks in methods which does not use throws class and does not have any in the original code. At the catch block a SER log is placed. For each rule that matches, the construct is inserted by the code parser. At run time the log messages are expected to be posted as messages to the log bus using log4x. The bus will process the message and direct to the output type configured.

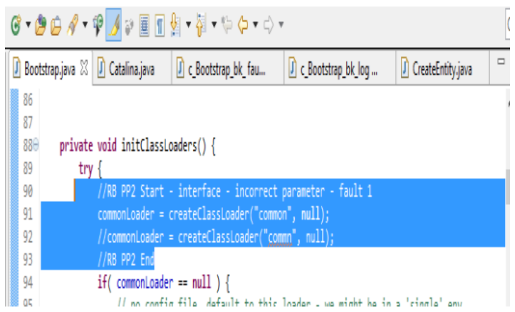
Each rule definition has information on event code, event type and where to insert the log code. Appropriate code transformation tool may be used here to avoid the complexities in handling input and output streams of data.

Table I. Rule Log code description

Rule	Rule Log code	Description
1	SST	Service Start
2	SEN	Service End
3	SER	Service Error
4	IST	Interaction Start
5	IEN	Interaction End
6	HTB	Heart beat
7	EXC	Unhandled Exception

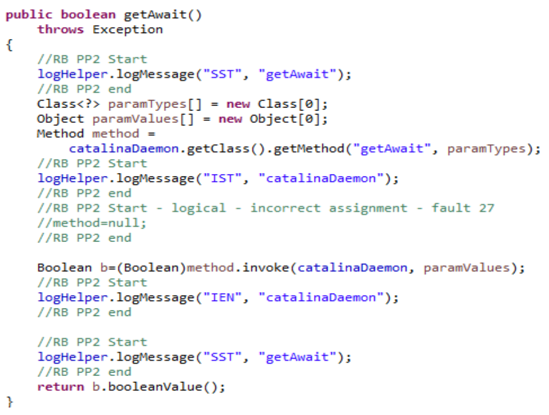
V. CASE STUDY

The case study undertaken was planned to be executed on Tomcat as it is commonly used for business purpose, it is Java technology based and functionality of Apache web server is a subset functionality of this. Similar approach should hold good for C# as well. The windows environment hosted all these components and the results were tested using manual injection of faults in selective classes. The environment used JDK 7 along with Apache Tomcat 7.0.x and log4j. Eclipse 4.4 Luna was used as the development and build environment. Thebootstrap process (the server start) has been taken to study the case. graph was generated for the bootstrap classes and logging code has been added. The added code was reviewed and any tool related errors were corrected. The code was executed and logs are saved.



```java
    private void initClassLoaders() {
        try {
            //RB PP2 Start - interface - incorrect parameter - fault 1
            commonLoader = createClassLoader("common", null);
            //commonLoader = createClassLoader("commn", null);
            //RB PP2 End
            if( commonLoader == null ) {
                // no config file, default to this loader - we might be in a 'single' env
```

Fig 2: Fault Injection sample



```java
public boolean getAwait()
    throws Exception
{
    //RB PP2 Start
    logHelper.logMessage("SST", "getAwait");
    //RB PP2 end
    Class<?> paramTypes[] = new Class[0];
    Object paramValues[] = new Object[0];
    Method method =
        catalinaDaemon.getClass().getMethod("getAwait", paramTypes);
    //RB PP2 Start
    logHelper.logMessage("IST", "catalinaDaemon");
    //RB PP2 end
    //RB PP2 Start - logical - incorrect assignment - fault 27
    //method=null;
    //RB PP2 end

    Boolean b=(Boolean)method.invoke(catalinaDaemon, paramValues);
    //RB PP2 Start
    logHelper.logMessage("IEN", "catalinaDaemon");
    //RB PP2 end

    //RB PP2 Start
    logHelper.logMessage("SST", "getAwait");
    //RB PP2 end
    return b.booleanValue();
}
```

Figure 3. Sample of logged code

The faults used in the study are designed based on the study work carried out by Duraes [6]. A sample logging and fault injection is shown in Figure 2 & 3. A series of such faults were injected into the code one at a time and logs were taken for each execution. The same faults were added to the as-is code (without proposed solution) one by one and logs were saved. Both of them are compared and the analyzed results are summarized in table 2 and figure 4.

VI. EXPERIMENTAL RESULTS

Table II. Fault detection metrics

Original Code			
Fault category	No. of Faults injected	No. Detected	No. Non-detected
Assignment	12	8	4
Algorithm	15	2	13
Checking	2	0	2
Interface	2	0	2
Code with Proposed Solution			
Fault category	No. of Faults injected	No. Detected	No. Non-detected
Assignment	12	11	1
Algorithm	15	14	1
Checking	2	2	0
Interface	2	2	0

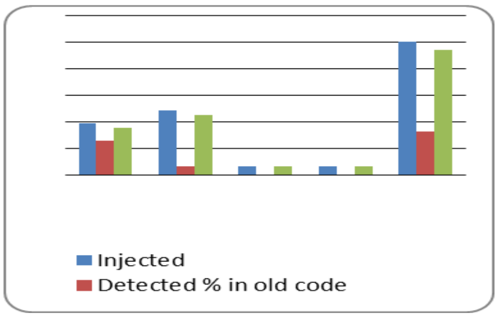


Fig. 4. Fault Detection statistics

The results indicate that the suggested approach reveals 94% of the faults overall whereas the original code detects 35% of the faults

VII. CONCLUSION

Most of the applications in today's world are written in object oriented languages. It is important to have a well defined approach to have sound defect detection capability of these applications. This work aims at improving the logging pattern for software applications. The rules and semantics are defined in XML format so that they can be handled with configuration alone. The approach best suits systems that are bigger in nature and also product kind of systems which continuously evolves and numerous dot release and patches could get in on a weekly basis. The study detects 94% of the faults injected and the approach proved to be successful. The study carried out is preliminary in nature and has to be scaled up using fault injection tools in future so that improved detection accuracy can be analyzed using high volume testin

REFERENCES

[1] Marcello Cinque, Domenico Cotroneo, Antonio Pecchia, "Event logs for the Analysis of Software Failures : A Rule based Approach" IEEE Transactions on Software Engineering Vol 39, no 6. June 2013

[2] K. Umadevi, S.Brintha Rajakumari, K.Ramya (2015), "Software Fault Detection and Diagnostic Techniques:A Review and Current Trends" International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE), 41-46.
http://depfind.sourceforge.net/
http://sourceforge.net/projects/logbus-ng/.
http://www.information-management.com/infodirect/2009_133/downtime_cost-10015855-1.html

[3] Duraes, J.A.; Madeira, H.S., "Emulation of Software Faults: A Field Data Study and a Practical Approach," SoftwareEngineering, IEEE Transactions on , vol.32, no.11, pp.849,867, Nov. 2006

[4] Ding Yuan,Jing Zheng, Soyeon Park, Yuanyuan Zhou, Stefan Savage" Improving Software Diagnosability via Log Enhancement", Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems , ACM New York, NY, USA ©2011

[5] Ariel Rabki, Wei Xu, Avani Wildani,Armando Fox, David Patterson and Randy Katz," A graphical representation for identifier structure in logs", Proceeding SLAML'10 workshop on Managing systems via log analysis and machine learning

[6] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko, " Failure prediction based on log files using Random Indexing and Support Vector Machines," J. Syst. Softw. 86, 1 (2013), 2-11

[7] P. Zoeteweij, R. Abreu, and A.J.C. van Gemund, "Software fault diagnosis," in IFIP Int'l Conf. on Testing of Communicating Systems: Hand-Outs for the Tutorial Day of TestCom / FATES. TartuUniversityPress,2007

AUTHOR'S PROFILE

K.Shanmugapriya, Assistant Professor, Department of Computer Science & Engineering, Bharath Institute of Higher Education and Research, Chennai, India

C. Geetha, Assistant Professor, Department of Computer Science and Engineering, Bharath Institute of Higher Education and Research, Chennai, India..

I.MaryLinda, Assistant Professor, Department of Computer Science & Engineering, Bharath Institute of Higher Education and Research, Chennai, India

D.Vimala, Assistant Professor, Department of Computer Science & Engineering, Bharath Institute of Higher Education and Research, Chennai, India