Software Application Logging: Aspects to Consider by Implementing Knowledge Management

Conference Paper · August 2016

DOI: 10.1109/OBD.2016.22

2 authors:

Monika Dávideková
Comenius University in Bratislava
39 PUBLICATIONS **135** CITATIONS
SEE PROFILE

Michal Greguš
Comenius University in Bratislava
48 PUBLICATIONS **213** CITATIONS
SEE PROFILE

Software Application Logging

Aspects to Consider by Implementing Knowledge Management

Monika Dávideková[1,2]
[1]Institute of Telecommunication, FEI STU,
[2]Vysoká škola manažmentu
both in Bratislava, Slovakia
e-mail: davidekova@ut.fei.stuba.sk

Michal Greguš ml.
Department of Information Systems
Faculty of Management, Comenius University
Bratislava, Slovakia
e-mail: Michal.Gregus.ml@fm.uniba.sk

***Abstract*— Standard workday today includes use of software applications that enable faster task processing and can do so for several tasks in parallel for us. Still, also the best applications can stop working or may not work correctly. For investigating such unintended malfunctioning of a software application, logging is very important. By examining application logs, software developers can find the cause and provide an update or patch with fixed bugs. On the other hand, logging can provide enormous amount of data that can flood the memory. That is why it is important not to log everything. The decision about logging (how to do logging of what data) is done during the software development phase. This paper describes research that has been focused on logging strategies in form of case studies of software development projects and in conclusion we give some guidelines for future decisions on logging for software development teams that may lead to logging improvement.**

Keywords- data; information; knowledge; logging; big data; knowledge management

I. INTRODUCTION

Information and Communications Technology (ICT) has caused revolutionary changes in the way people work, communicate, learn, spend time, and interact [1]. In the last couple of decades no technology has had such a global impact as ICT [2]. ICT bolstered productivity more effectively than any other earlier technologies [3]. It provides the base for computer applications to support business processes [4]. Mobile cellular and telecommunication technology are the dominant technology drivers and hence have relatively high contributory power to support the long-run economic growth [5].

In today's business life, several workers interact with software applications to carry out working tasks on a daily basis and to benefit from the possibilities offered by the technology. Working with text and spreadsheet processors, communicating via mobile phones, using electronic mail service and several others became a substantial part of working, academic and private life. Companies are using IT solutions to support their business operations that allow to process data on automated way and much faster. IT is not a mere enabler for business activity anymore [6]. ICT and IT drive business strategy, open new markets and possibilities.

Software is intended to automate a manual process or tool so that physical artifacts are replaced by more malleable and versatile virtual artifacts [7]. Companies are executing business activities to produce company products. People spend a lot of time on their computers performing repetitive tasks that could easily be automated, thus saving time and eliminating tedium [8]. Although computers are supposed to excel at performing repetitive tasks, most users are unable to take advantage of this capability because they do not know how to program [9]. Therefore, they contract a software company that specializes in the development of software applications. Software companies develop software products that perform automated tasks. IT solutions developed by software programmers are highly tailored to the demands of the customer and specialized for the customer.

Still, no program is perfect as there still may be external factors that were not considered during the development phase and may lead to program malfunctioning or not working. To be able to find the root cause for such an undesired behavior, the application is usually capturing its process stages and variables and recording it into a specified destination. Applications often need to log various messages representing conditions or events [10] and record them to a log file. However, in the practice logs are often missing, they omit critical details, or they have no standard form or content [11].

As log files have been originally meant for debugging purposes [12] they may contain a large amount of data, in other words, the size of the log file can become very large very fast. The purpose of log entries is also to be able to analyze the inputs and calls that lead to such an application failure as the system on which the application run may change with time too and hence, simulation results may not be completely reproducible [13].

Another aspect to consider during software design phase is that logging may take time [14]. Therefore, there is natural design decision what to log or how to log.

This paper focuses on investigating how development teams in business practice are managing the challenges of logging and which strategies they apply. This paper is organized as follows: section II analyzes aspects of logging from literature review. In section III the researched case studies are described and research findings are discussed. Conclusion summarizes the outcomes of this research.

978-1-5090-4054-4/16 $31.00 © 2016 IEEE
DOI 10.1109/OBD.2016.22

II. LOGGING ASPECTS

A "log" is defined as a record of performance, events, or day-to-day activities by Meriam-Webster Dictionary [15] and as a regular or systematic record of incidents or observations by Oxford Dictionary [16]. This definitions are in compliance with the application logging that is used today in software development and maintenance where applications often need to log various messages representing conditions or events [10].

A log record is designed to help to analyze malfunctioning of the application's behavior and to identify the part of application source code that leads to such an undesired behavior. Log files, originally meant for debugging purposes [11], may help to investigate what happened and how did it happen even without the repeated application starting if these logs include all needed information in their log entries. However, this is often not true, unfortunately. In the practice logs are often missing and omit critical details [11] or in a lot of cases, application developers do not log much [17]. Therefore, it is important to include logging in the design of software applications.

When including logging in the software application development, it is important to decide upon what to log and how to log. The best logs tell you exactly what happened, when, where, and how [11]. Every log entry should, if possible, log what happened, when it happened, who triggered the event, and why it happened [17]. Resulting from these guidelines, a log entry should if possible always answer following questions:

1. What did happen?
2. When did it happen?
3. Where did it happen?
4. Who triggered the event?
5. Who was involved?
6. Why did it happen?
7. What caused it to happen?

These few questions should always be answered if possible in logging. The answers to some of these questions are straightforward.

The answer to the first question should indicate, which type of occurrence happened (warning, error or information) and what was affected by the occurrence (system, component, data resource, user account). It should also indicate the state of the occurrence, e.g. if something failed to happen.

The second question is answered by the time stamp of the occurrence. For distributed applications it is helpful to include also a time zone reference. It is helpful to indicate whether the time stamp is given upon server time with a time zone as the actual time may underlie a change (e.g. the server is not switching between winter and summer time what might lead to investigation delays).

The third question tells us which system, application, class was affected and partially could indicate the cause, e.g. when the database was not accessible that caused the application failure. The answer to this question should contain the initiator system and the target system, e.g. that the application tried to write to a database.

The fourth question is answered by the user name that tell us who actively cause it to happen, in other words, whose actions lead to the occurrence.

The fifth question tells us if the happened event affected only the one user who caused it, or if it involved more end users.

The sixth question tells us the reason of the occurrence, e.g. incorrect password, settings not set up, a choice not selected, etc.

The seventh question tells us which type of event happened, in other words, which action lead to this occurrence (login, calculation start, etc.).

However, the logging implementation is not always that simple as it may seem. The strategy for logging depends on the application environment and the end user for who the application is designed for.

Logging all the answers to the set of questions might denote a large amount of log entries to be included in a log file. The size of the log file may affect the application operation, if too large amount of data are being logged and thus, the memory can become flooded with logging data. It is also important to determine upfront, for which period of time the log files will be maintained and deleted afterwards. Without deletion of old logging files, the storage may become overfilled with historical no longer relevant data and lead to memory issues for the application. This is similar when logging on the computer for standalone applications.

These log entries will represent data from which the log reader will try to extract information about what is wrong and to turn it into knowledge about how to prevent it and how to fix it. It is important to thing about the logging strategy in advance, during the design phase. This is also expressed by Tuomi [18] who stressed the importance of knowledge for subsequent data collecting: Knowledge is needed before data are collected and indeed, it determines which of these data to store. In other words, it is important to decide upfront which data to include in the log entries that may help the log reader. To log everything (every method including getters and setters) may lead to extreme large useless log files and to memory overflow.

ICT can be seen as a crucial enabler for knowledge transfer [3]. This means also to know how to transfer to knowledge, in other words, logging can be seen as knowledge transfer from the application to the developer in form of logged data in the log file. Zitat?

Another aspect is, that logging also takes some time [14] and therefore, it is important to decide when to log: whether the method or action shall be started prior logging, parallel to it or after it. Logging prior the method execution may lead to delays and logging after execution may not include the input data in the log entry.

There are also aspects that shall not be included in the log files as passwords or other sensitive data [11].

III. METHODOLOGY

The research carried out consisted of literature review and of case studies of projects for software application development. In our sample we included software application development projects in various programming

languages on a voluntary basis in form of interviews and observations focused on the use of logging implemented in the particular software application.

The sample consisted of Java and C# applications where the Java software application projects included web applications and the C# software application projects contained standalone application. This was not intended, however, it is a very important fact that explains a large amount of differences in the research findings.

The number of projects investigated consisted of 30 projects devoted to software application development and was unequally distributed between Java and C# (22 Java projects, 8 C# projects).

TABLE I. INVESTIGATED SOFTWARE DEVELOPMENT PROJECTS

Software Development Projects included in Case Study		
Total	***Java***	***C#***
30	22	8

A. Investigated Java Projects

The Java projects included development teams from a small team size (1 to 2 developers) to a very large size (20 to 30 developers) and the designed end user varied from external customers to company employees from other department. However, the end users were no programmers and the designed end user factor had no influence on used logging approach in the Java projects. No real connection to the decision about the logging was found during the investigation at least.

The character of the investigated projects varied too. One large project was designated to create a sophisticated complex system of order processing and maintenance. The smaller java projects were designed to create orders or to merely display stages of orders or transactions with relevant information associated with the given transaction or order. Although the design scopes differentiated largely, we could generalize those into wide public where the customers do not need to be specialized for using these applications or to dispose with large background information. Therefore, the varying end user differences had only a very little impact on the decision upon the logging strategy. The developers considered the end user as mere receiver of information displayed or produced by the software application. The software application developers included in projects were the targeted readers of the produced application logs and this was the most important factor that influenced the logging used in those applications.

Later, during the maintenance period of some of these projects, this caused that the developers were always contacted when an application failure was detected.

The Java projects used Log for J (Log4J) from Apache [19], Simple Logging Facade for Java (SLF4J) [20] and LogBack [21]. None of investigated projects used the native built-in Java.Util.Logging package [22]. Further Java logging frameworks and strategies can be found nicely summarized in [23]. The below depicted table mirrors the unequal distribution of logging frameworks used by investigated projects.

TABLE II. JAVA LOGGING FRAMEWORKS USED

Logging Frameworks Used in Java Projects			
Total	***Log4J***	***SLF4J***	***LogBack***
22	6	8	8

The software development teams were switching from Log4J over SLF4J towards LogBack logging framework due to the overhead and constraints of Log4J. The youngest projects were all using LogBack, where the reason was smaller overhead and bigger possibilities developers

Java projects used a logger set up in a XML file (valid for all three used java logging frameworks), included it as a private field in each class and logged their messages.

The logging was done directly by calling the log method on the logger instance and also done via Aspects before and after a method was executed. The second approach was used very rarely, though.

The developers were logging the user name of the logged user once, at the time of the application start or his/her successful authentication towards the application, the time stamp, the name of the java class that was executing an operation and the stack trace of an exception when occurred (in the try catch blocks). The directly called log calls on the logger contained mostly pure business logic stages (e.g. user authorized, rights available, new projects created etc.) Still, the logs included a large amount of information which was logged by the application itself by the used spring framework and not by the developers. This was causing the files getting extremely large within few days. The logging files were appended each time a new transaction started. The maximum file size was given in the XML settings and when it was reached, the logging framework started to log into new file without the intervention of the software developer. This setting kept the number of log files down.

However, the amount of logged data, in other words, the number of log entries, was very large and this was causing issues with memory space. The production memory and storage became overfilled pretty fast what resulted in several extensions of storage of given servers and in close monitoring of the available space during the operation of these applications. The old logs were deleted manually in periods of few months or at the point of low free memory space notification of the monitoring service to clean the filled space and to avoid production break down.

During the maintenance of the developed applications, the logs were analyzed by the software developers and by the environment administrators for the root cause analysis. The identification of the causes for the application undesired behavior was not always possible, often due to late notification (when the logs were already deleted) or to the fact, that the log entries have not enabled to simulate the stage of the application that lead to the wrong behavior of the application. The log files have helped only in a limited number of cases, mostly at time when the single sign on has

not worked or when the login credentials were wrong or when a database was not accessible.

Following the seven questions defined in previous chapter, the java logging used in investigated projects answered these questions in following manner as captured in the table below.

TABLE III. INVESTIGATED SOFTWARE DEVELOPMENT PROJECTS

Question to Answer in the Logging	Approach used in investigated java projects
What did happen?	Stack trace of an exception logged Direct log messages informing that a method was successfully finished
When did it happen?	Server time stamp
Where did it happen?	Java class name, called method
Who triggered the event?	User name
Who was involved?	User name
Why did it happen?	Stack trace of an exception logged Direct log messages informing that a method was successfully finished
What caused it to happen?	Stack trace of an exception legged

As it can be seen, the developers were logging mostly direct messages or exception, but the input parameters that may have led to such occurrences were not logged and the log entries included in those log files were only partially helpful for the root cause analysis.

The log file analysis have shown that even some credentials were logged in plain form. This was true for older projects and improved in the newer projects where the software developers were implementing hash functions for authentication.

B. Investigated .NET Projects (C#)

The C# projects consisted of standalone software applications that were distributed to a limited number of specific users. In other words, the users of those applications were scientific specialized engineers working for the company/companies. Here the designed end user factor had a strong influence on the decision of used logging approach. As the end user denoted a specialized engineer with a lot of knowledge about the calculations of the provided software, the engineers were able to deliver also the logs with corresponding entries what allowed another logging approach.

The project sizes of these software development projects were considerably small, where the software developer teams consisted from one to 10 developers per project.

In .NET there are several logging frameworks too [24], where to the most famous logging frameworks today belong Log for .NET (Log4Net) from Apache [25] and .NET Log (NLog) [26] which I becoming popular nowadays. Those frameworks are alike the frameworks known in Java, they have to be called directly. Alike the aspect programming in Spring and logging before and after the actual method execution provided by annotation is also possible in C# and offered by PostSharp [27] that allow logging of all input parameters and output parameters of a method too.

The Log4Net and NLog are being set up through a XML file just like their analogic counterparts in Java. PostSharp allow the aspect programming and setting up through a wizard.

Despite all these possibilities, the software developer teams of participated investigated project decided to user their own simple logger for logging. Logging is modelled as a separate, singleton class [28] that is included in the project and called.

TABLE IV. .NET LOGGING FRAMEWORKS USED

Logging Frameworks Used in .NET Projects	
Total	***Own Logger***
8	8

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

The logging was triggered at each application start logging the user name of the authorized user. As the developed applications were developed and used inhouse the by specialists, the developers excluded unnecessary logging stages and included logging of stack traces of occurred exceptions and logged solely some stages of calculations or methods that the developers considered to be very important.

Developers did consider the knowledge of the end users and their cooperation in case of an application failure that can be solved fast in common cooperation with the end user.

The own implemented Logger disposed of settings set up in the own logger class and those settings were possible to set up anew in the application of course. Thus, the logger was working automatically after the including in the application project. Using other named frameworks like Log4Net or NLog would require a XML settings file.

The logging was done solely by direct calls of the log method on the logger singleton instance. No aspects were used for logging and no before and after logging of a method were used. As the methods were scientific calculations with extreme large input data containers, those were not logged in the log files, only the missing parts (when an exception occurred) were logged in order to prevent any mal functioning behaviors by precise scientific calculations.

The log entries included the time stamp and the caller method name of the executing method where the exception occurred. The developers also logged the calculation settings, however, no input data was logged.

The directly logged messages included stages of the business logic process which were already processed like which step of the calculation has been finished already and with which stage and why if the status was not finished.

The logs of the developer teams in C# did not include a large amount of information which was logged by the application itself (by the Prism framework) and thus the amount of the logged data was considerably small. The

logger created a new logging file for each application start what kept the log files small. The log files were done in the temporary directories of the end users and deleted by the system by memory issues. The applications were started each single day anew or more times per day what caused the files to be kept very small.

During the maintenance period of the developed applications, the software developer teams were able to identify the causes of the failures very fast as the stack trace was always logged in the log file and the log file was not overfilled with unnecessary data describing solely the application life stages. The developer teams kept the logs very simple and short and were able to fix their applications very fast. This was really surprising if compared to the little data that was logged.

In these cases, no plain passwords were logged as the authorization was solved on another way using the knowledge that all the end user are internal employees of given company/companies.

Following the 7 questions defined in previous chapter, the C# logging used in investigated projects answered the defined questions in following manner as noted in the table below.

TABLE V. INVESTIGATED SOFTWARE DEVELOPMENT PROJECTS

Question to Answer in the Logging	Approach used in investigated .NET projects
What did happen?	Stack trace of the logged exception Direct log messages informing which method was performed and with which stage
When did it happen?	Time stamp with the time of the computer operating system
Where did it happen?	Caller method name, exception
Who triggered the event?	User name
Who was involved?	Triggering component and user name
Why did it happen?	Stack trace of the logged exception Direct log messages informing which method was performed and with which stage
What caused it to happen?	Stack trace of the logged exception

C. *Comparison of the research findings*

As can be seen, the Java and C# development teams chose completely different strategies for their logging. The C# development teams used the advantage of the knowledge of the end user that helped them to develop their own logger and their own logging process.

The Java developers were using the logging frameworks and the possibilities offered by implemented Spring framework with aspect logging. Although, the logs were huge in comparison to the C# logs, the logged log entries were not always helpful in the root cause analysis and lead more to the memory overflow than to the bug documentation.

Here, we can conclude, that the C# developers used the approach of Tuomi [18] who says that knowledge is necessary to determine which data to collect with "data emerges only after we have information, and that information emerges only after we already have knowledge" [18], in other words, what is really worth to log and may help to further analysis in search of a bug.

Java developers followed the Spiegler approach with his "Yesterday's data are today's information, and tomorrow's knowledge" [29] that tells us that data turns into information and that can transform into knowledge after understanding.

This research findings are showing that it is important to think about the logging already in the time of the application development and to consider the answers to mentioned questions and that it is not always the best way to log everything possible even though, we would expect it.

The propositions which we would conclude are:

1. Knowing the end user may help to reduce unnecessary logging.
2. Designing the logging the way that a support employee will be able to retrieve the necessary information out from the log file may prevent unnecessary involving of the developers in the production support issues.
3. Answering of the 7 W questions in the log entry may help in the root cause analysis of the bug of the application:
   a. What did happen?
   b. When did it happen?
   c. Where did it happen?
   d. Who triggered the event?
   e. Who was involved?
   f. Why did it happen?
   g. What caused it to happen?
4. Viewing the log entries from the point of view of a maintainer may improve the log entries and shorten the root cause analyses.

IV. CONCLUSION

This paper focused on the logging topic and investigated approaches that are used in the business practice in software development teams. The logging aspects were discussed identifying the questions that are important to be answered in logging. Issues caused by the log entry amount were identified and discussed too.

The research carried out in form of case studies of voluntary participated projects devoted to software application development were described in further details analyzing the used logging approaches including the chosen logging framework. The log entries were analyzed and compared with the identified logging question to show where the entries match these questions and to identify room for improvement.

The case studies included a comparison of observed discrepancies of investigated projects and provided a discussion about the causes of these differences. The case studies included also observations from the maintenance of

those developed applications and the impact of logged data for the root cause search.

The last section provides propositions that were identified as important during this research and that may lead to improvement of the logging strategy used by development teams in the practice that has shown room for improvement.

ACKNOWLEDGMENT

This research has been supported by the Department of Information Systems, Faculty of Management, Comenius University in Bratislava, Slovakia.

We would like to thank to all software development teams who decided to join this case study focused on logging and provided us with the necessary information and enabled this research to be carried out.

REFERENCES

[1] D. W. Jorgenson and K. M. Wu, “The ICT revolution, world economic growth, and policy issues,” Telecommunications Policy, vol. 40, no. 5, May 2016, pp. 383-397, doi:10.1016/j.telpol.2016.01.002.

[2] N. R. F. Al-Rodhan, Information and Communications Technology (ICT): The Politics of Emerging Strategic Technologies. London, UK: Palgrave Macmillan UK, 2011.

[3] M. A. Hidalgo Pérez, J. M. O'Kean Alonso and J. Rodríguez López, “Labor demand and ICT adoption in Spain,” Telecommunications Policy, vol. 40, no. 5, Aug. 2015, pp. 450-470, doi:10.1016/j.telpol.2015.07.004.

[4] M. Broadbent and P. Weill, “Management by maxim: how business and IT managers can create IT infrastructures, “Sloan Management Review, vol. 38, no. 3, Nov. 1997, pp. 1-31.

[5] R. R. Kumar, P. J. Stauvermann and A. Samitas, “The effects of ICT* on output per worker: A study of the Chinese economy,“ Telecommunications Policy, vol. 40, no. 2-3, July 2015 pp. 102-115, doi:10.1016/j.telpol.2015.06.004.

[6] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana and G. Rackham, "Impact of service orientation at the business level," IBM Systems J, vol. 44, no. 4, 2005, pp. 653-668, doi: 10.1147/sj.444.0653.

[7] C. Potts, "Invented requirements and imagined customers: requirements engineering for off-the-shelf software," Proc. IEEE Int. Symp. Requirements Engineering 1995, IEEE Press, Mar 1995, pp. 128-130, doi: 10.1109/ISRE.1995.512553.

[8] M. Gervasio and K. Myers, "Question asking to inform procedure learning," Proc. AAAI-08 Workshop on Metareasoning: Thinking about Thinking, AAAI, July 2008, pp. 68-75.

[9] A. Cypher, "Eager: Programming repetitive tasks by example," Proc. SIGCHI Conf. Human Factors in Computing Systems, ACM, May 1991, pp. 33-39, doi:10.1145/108844.108850.

[10] R, Chinta and S. Das, “Application server message logging,” US Patent US6879995 B1, Apr. 2005.

[11] A. Chuvakin and G. Peterson, "How to Do Application Logging Right," IEEE Security & Privacy, vol. 8, no. 4, July/Aug. 2010, pp. 82-85,. doi: 10.1109/MSP.2010.127

[12] K. R. Suneetha and R. Krishnamoorthi, "Identifying user behavior by analyzing web server access log file," IJCSNS, vol. 9, no. 4, Apr. 2009, pp. 327-332, doi:10.1.1.586.46.

[13] S. Narayanasamy, C. Pereira, H. Patil, R, Cohn and B. Calder, “Automatic Logging of Operating System Effects to Guide Applicationlevel Architecture Simulation,” Proc. Joint Int. Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '06/Performance '06), ACM, June 2006, pp. 216-227, doi:10.1145/1140277.1140303.

[14] L. Alvisi and K. Marzullo, "Message logging: pessimistic, optimistic, causal, and optimal," Proc. 15th Int. Conf. Distributed Computing Systems, IEEE Press, June 1995, pp. 229-236, doi: 10.1109/ICDCS.1995.500024.

[15] Merriam-Webster Dictionary, “Log,”. Retrieved April 2, 2016, available in internet: http://www.merriam-webster.com/dictionary/log

[16] Oxford Dictionary, “Log,” retrieved Apr. 2, 2016, available in internet: http://www.oxforddictionaries.com/definition/english/log

[17] R. Marty, ”Cloud application logging for forensics,” Proc. 2011 ACM Symp. Applied Computing, ACM, March 2011, pp. 178-184, doi:10.1145/1982185.1982226.

[18] I. Tuomi, “Data is more than knowledge: Implications of the reversed knowledge hierarchy for knowledge management and organizational memory,” Proc. 32nd Annual Hawaii Int. Conf. Systems Sciences, 1999 (HICSS-32), IEEE Press, Jan. 1999, pp. 1-12, doi: 10.1109/HICSS.1999.772795.

[19] Apache Log4j 2, Java Logging, retrieved Apr. 3, 2016, available in internet: http://logging.apache.org/log4j/2.x/

[20] Simple Logging Facade for Java (SLF4J), Java Logging, retrieved Apr. 3, 2016, available in internet: http://www.slf4j.org/

[21] Logback Project, Java Logging, retrieved Apr. 3, 2016, available in internet: http://logback.qos.ch/

[22] Java.Util.Loging Package, Java Logging, Retrieved April 3, 2016, available in internet: https://docs.oracle.com/javase/7/docs/api/java/util/logging/packagesummary.html

[23] Java Logging Tools and Libraries, .NET Logging, retrieved Apr. 3, 2016, available in internet: http://www.java-logging.com/

[24] .NET Logging Tools and Libraries, .NET Logging, retrieved Apr. 3, 2016, available in internet: http://www.dotnetlogging.com/

[25] Apache log4net project, .NET Logging, retrieved Apr. 3, 2016, available in internet: https://logging.apache.org/log4net/

[26] NLog, Flexible & free open-source logging for .NET, .NET Logging, retrieved Apr. 3, 2016, available in internet: https://logging.apache.org/log4net/

[27] PostSharp Diagnostics: Logging and Tracing, .NET Logging, retrieved Apr. 3, 2016, available in internet: https://www.postsharp.net/diagnostics/net-logging

[28] S. Clarke, W. Harrison, H. Ossher and P. Tarr, “Subject-oriented design: towards improved alignment of requirements, design, and code,” ACM SIGPLAN Notices, vol. 34, no. 10, Oct. 1999, pp. 325-339, doi:10.1145/320385.320420.

[29] I. Spiegler, “Knowledge Management: A New Idea or a Recycled Concept,” CAIS, vol. 3, no. 4es, June 2000, Available in internet: http://aisel.aisnet.org/cais/vol3/iss1/14.