IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-2 E-ISSN: 2347-2693

Applying Dependency Injection Through AOP Programming to Analyze the Performance of OS

Jatin Arora[1]*, Jagandeep Singh Sidhu[2] and Pavneet Kaur[3]

[1*,2,3] *Computer Science Department, Chitkara University, India*
jimjatin@gmail.com, jagandeep.sidhu@chitkara.edu.in, pavneet1812@gmail.com

www.ijcaonline.org

Received: Feb /6/2015 Revised: Feb/14/2015 Accepted: Feb/23/2015 Published: Feb/28/ 2015

***Abstract*—** Operating systems are very inflexible towards modification of already existing functionalities such as security, dynamic re-configurability, robustness etc. In such functionalities if need arises for any enhancements then it effects on large fractions of the code. Thus results in difficult to implement. Such functional enhancements in any component of the OS that affect large fractions of the program code, are often called crosscutting concerns. Such cross-cutting concerns can be solved by the new emerging extension to object oriented paradigm i.e. Aspect Oriented Programming (AOP). The main idea in AOP is the programmer's ability to affect the execution of core code by writing aspects. Aspects are pieces of code that are run before, or after core function for which aspect is written. For example logging is a good example of using aspects. To log all function calls the programmer simply needs to define a logging aspect that is executed before and after each function call in the program. In this work aspects are introduced in the Operating System for implementing various concerns and analyzing the performance based on various metrics.

***Keywords*—**Aspect; Pointcut; Before; After; Aspectj; Cross Cutting; Concerns; Dependency

I. Introduction

All standard paper components have been specified for three reasons: (1) ease of use when formatting individual papers, (2) automatic compliance to electronic requirements that facilitate the concurrent or later production of electronic products, and (3) conformity of style throughout conference proceedings. Margins, column widths, line spacing, and type styles are built-in; examples of the type styles are provided throughout this document and are identified in italic type, within parentheses, following the example. Some components, such as multi-leveled equations, graphics, and tables are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow Computer devices can perform operations in fractions of seconds and without any error which normal human require his lifetime to do. But computers as such are useless without the software to run on them. And all the software's are useless too without the Operating system which is an important part of computer system. Operating system is responsible for all the other programs to run on the computer. Various programming strategies have been applied for the development of operating system. Originally operating system was developed using assembly language or C language. Using C programming in the development of Operating system caused many problems, because C is a procedural language. Programming practices were not modular in the early computer's age and they weren't easily modifiable, but with the advent of modern computers, modular configuration is possible. The performance of an OS is not as important as it was in the past because now a day's computers are more advanced and powerful. The priority now is given to security and stability of OS rather than on its processing time and memory usage. Moreover, modern operating systems are highly complex to develop them in low-level programming languages, so a solution proposed to this problem is to use a high-level language.

Many new high level languages such as C++ and other object oriented programming languages have been used for improving the development of OS. Various research operating systems have been developed using high level language but still operating systems face problem with modularity. Complex interactions due to dependencies between the modules of the system threat module's boundaries and make them highly susceptible to errors. Security, authentication checks, exception handling, statistics, dynamic re-configurability, console are the major modules of any operating systems but if any modifications are required in these modules then it effects on the large fraction of code which is difficult to implement. The operating system used for carrying out this task is NachOS and it is used for research purposes. It is a rather small and simple operating system but still has all the important features of a real operating system.

Although there is progress in the field of separating cross cutting concerns, but still lot of work can be done in the operating systems cross cutting concerns. This paper considers the logging, security and console handling as the major concerns to be resolved by implementing them with AOP as well as with that of Java. For statistical measurement of the code the LOC metric, i.e. Lines Of Code is appropriate and simple metric. Although it is not very accurate measure of code complexity, it can still be used to compare two different implementations. Execution time could also be a statistical metrics for analyzing the execution time of aspect implementation as well as without aspect implementation.

II. Aspect Oriented Programming

The motivation and need for AOP arises from the short comings of traditional programming strategies. Development of software using AOP addresses the fall backs which arise due to cross cutting concerns in traditional approaches [17]. Generally large scale applications consist of requirements which can be achieved collectively by various modules. Instead of modularization, developers are required to make module which have various functionalities i.e. mixed goals. Example of one such module or concern is logging which is distributed throughout the whole software. Another important concern is security. Since, security functionality is required in most of the application so this module is also of the utmost concern. Also, if some new functionality is added in the system then security can also be asked to be imposed on them too. Further, the concept of Dependency Injection along with AOP is used in the present thesis as Dependency Injection when implemented using AOP helps to simplify the design of software applications. Dependency injection was introduced by Martin Fowler. It is a software design strategy that implements the principle of dependency inversion control. It is a technique for reducing the dependency among the components which in turn results in reusability of the components. Reducing dependencies from the system helps in decreasing the coupling among the various components of the system so that a better framework is achieved by separating cross cutting concerns from the main component into a different unit. The key responsibilities of DI are as follows:

- To determine which objects need configuration.
- To determine dependencies among the objects.
- To find the objects that satisfies similar dependencies.
- To configure the objects with respect to its dependencies.

A. AspectJ

AspectJ in AOP allows programmers to have the advantage of modularization for cross cutting concerns that are present in almost every part of software. In OOPs like C++ or Java, class is considered as modular unit. Similarly in AOP aspects provide the same functionality to the cross cutting concerns which provides functionality to more than one class. Program in AspectJ is compiled with its compiler and is then run with a runtime library. AspectJ is an asymmetric aspect-oriented language in that it distinguishes between core and crosscutting concerns, specifying them differently. Core concerns continue to be implemented in pure Java, modularized within classes and methods. Their implementation is referred to as the base program. Crosscutting concerns are implemented in aspects, using an extended syntax of Java. The aspects and base program are composed together to produce the complete program.

III. IMPLEMENTATION OF CROSS CUTTING CONCERN IN NACHOS USING AOP AND JAVA

The NachOS simulator provides the following components. Each component has its own functionality for providing the basic components required for functioning of an operating system.

1. Time	2.Disk
3. Console	4.Network
5. MIPS Processor	6. Interrupts

A. Cross Cutting Concerns in NachOS Units

In the present paper work is focused on the cross cutting concerns which are security, authentication, console handling. In addition to this logs for all the concerns are recorded and maintained for inspection and managing purposes.

B. SECURITY CONCERN

One of the cross cutting concern i.e. security is provided by the authenticate function. The function is acting like a join point in the machine simulation code. This join point acts as a place where advice can be injected on the basis of their matched criteria. The advantage of using security aspect i.e. authentication using AOP is that the new authentication features can be easily introduced in the present code of security. Every module that is requiring security for its execution, the aspect implementation is desired instead of calling a function for the security action.

C LOGGING CONCERN

Logging is the concern that is required for maintaining the logs of all the activities, exceptions occurring in the system, security breach actions, system resource access pattern and many more events. Logging is the major cross cutting concern in any application but the most prevalent application that requires logging is OS. The logs are maintained for the verification of the user’s task as well as for the improvements in the systems for the increased performance. Similarly in the NachOS, logging of various tasks performed, security logs, system usage time logs and many more concerns are cross cutting concern. Logging in NachOS is done by placing join points in the NachOS’s source files. Dependencies can be easily injected in the files by associating proper advice on the join points. In NachOS.machine.java file, a log function is defined and called at various functions where the need of log is to be maintained.

D. CONSOLE HANDLING

Displaying data on the console of the machine is directed through the standard output stream associated with the display device and getting data from the user is done by the input device (keyboard) connected via standard input stream. On this system, whenever the individual keystroke occurred on the simulated terminal, simulated terminal sends the corresponding code associated with the character for display.

Fig.1. and Fig.2. shows how an Operating system would implement logging using conventional techniques. This is where AOP comes into the scenario. Using the AOP paradigm, none of the core modules will contain the code of logging services—they don't even need to consider the presence of logging facility in the system. AOP implementation of the above mentioned logging functionality is shown in the fig.2. The logic for logging is now coded inside the logging module and logging aspect; clients are no longer needs to add any code for logging functionality. The crosscutting concern i.e. logging is now directly mapped to only one module—the logging aspect. With such modularization, any modifications to the crosscutting logging concern would going to affect only the logging aspect, separating the clients completely.

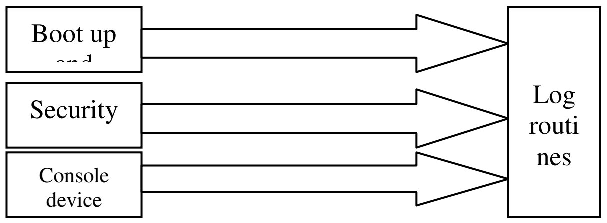


Fig.1. Tangling of Log routine with other modules

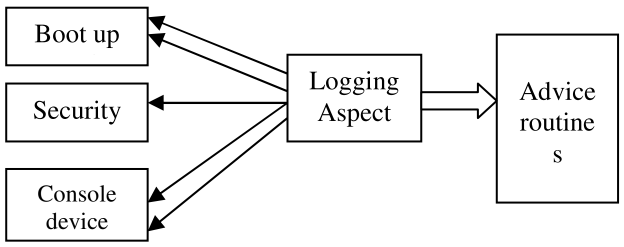
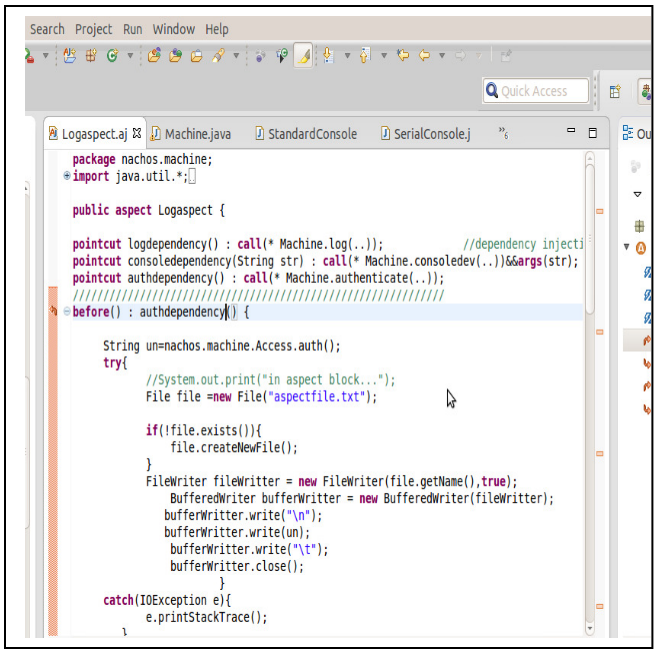


Fig.2. Aspect Implementation of Log

The pointcuts defined in the aspect of log are shown below in the figure. The pointcuts used for the NachOS machine file are the conditions that are going to check for the available join points in the machine code of NachOS. The first pointcut that is used for log maintenance is log dependency. In this pointcut, the wildcard is matching the join point on the basis of method call i.e. log() in the Machine class with any number of arguments.

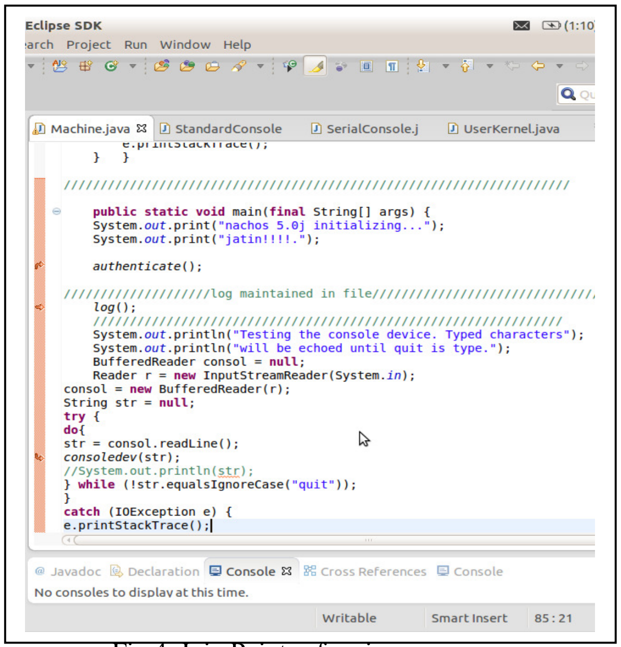


```
package nachos.machine;
import java.util.*;

public aspect Logaspect {

pointcut logdependency() : call(* Machine.log(..));          //dependency injecti
pointcut consoledependency(String str) : call(* Machine.consoledev(..))&&args(str);
pointcut authdependency() : call(* Machine.authenticate(..));
//////////////////////////////////////////////////////////////
before() : authdependency() {

     String un=nachos.machine.Access.auth();
     try{
            //System.out.print("in aspect block...");
            File file =new File("aspectfile.txt");

            if(!file.exists()){
                file.createNewFile();
            }
            FileWriter fileWritter = new FileWriter(file.getName(),true);
                BufferedWriter bufferWritter = new BufferedWriter(fileWritter);
               bufferWritter.write("\n");
               bufferWritter.write(un);
                bufferWritter.write("\t");
                bufferWritter.close();
                        }
     catch(IOException e){
            e.printStackTrace();
```

Fig.3. Pointcuts of cross cutting concerns

```
  pointcut logdependency() : call(* Machine.log(..));
                                        //dependency injection//
pointcut consoledependency(String str):call(*
Machine.consoledev(..))&&args(str);
pointcut authdependency() : call(* Machine.authenticate(..));
```

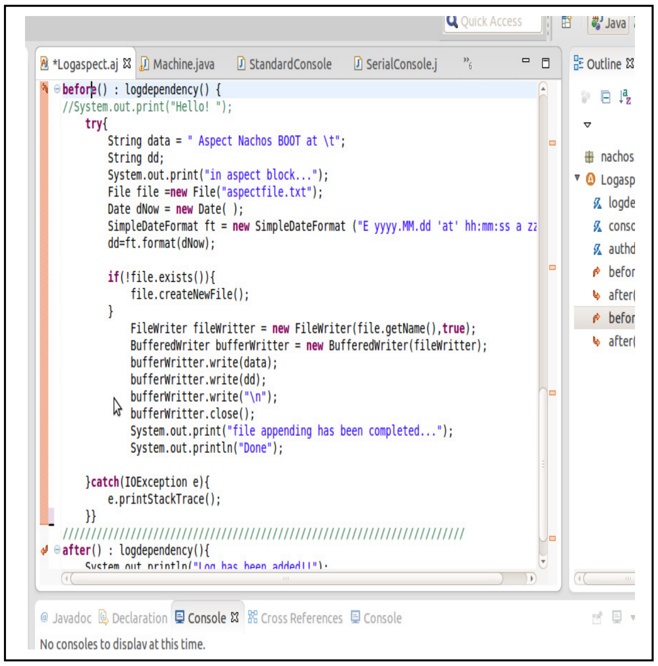
The action to be performed and decision making part of the crosscutting concern is the advice. It also helps to define "what to do." Advice is a method-like construct that provides a way to express crosscutting action at the join points that are captured by a pointcut.



```
          e.printStackTrace();
    }   }

////////////////////////////////////////////////////////////////////////

    public static void main(final String[] args) {
    System.out.print("nachos 5.0j initializing...");
    System.out.print("jatin!!!!.");

    authenticate();

////////////////////log maintained in file/////////////////////////////
    log();
    ///////////////////////////////////////////////////////////////
    System.out.println("Testing the console device. Typed characters");
    System.out.println("will be echoed until quit is type.");
    BufferedReader consol = null;
    Reader r = new InputStreamReader(System.in);
consol = new BufferedReader(r);
String str = null;
try {
do{
str = consol.readLine();
consoledev(str);
//System.out.println(str);
} while (!str.equalsIgnoreCase("quit"));
}
catch (IOException e) {
e.printStackTrace();
```

```
No consoles to display at this time.
```

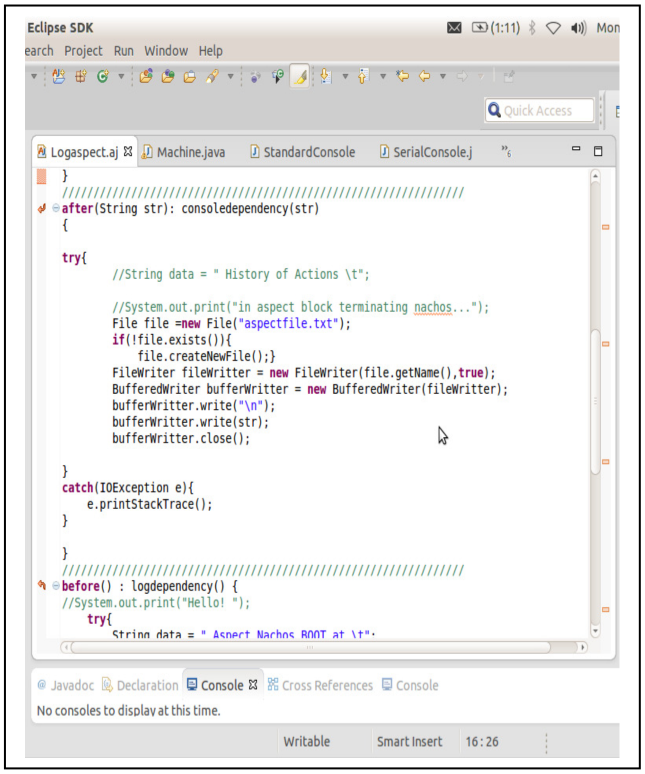
Fig.4. Join Points of various concerns



```
before() : logdependency() {
//System.out.print("Hello! ");
    try{
        String data = " Aspect Nachos BOOT at \t";
        String dd;
        System.out.print("in aspect block...");
        File file =new File("aspectfile.txt");
        Date dNow = new Date( );
        SimpleDateFormat ft = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zz
        dd=ft.format(dNow);

        if(!file.exists()){
            file.createNewFile();
        }
            FileWriter fileWritter = new FileWriter(file.getName(),true);
            BufferedWriter bufferWritter = new BufferedWriter(fileWritter);
            bufferWritter.write(data);
            bufferWritter.write(dd);
            bufferWritter.write("\n");
            bufferWritter.close();
            System.out.print("file appending has been completed...");
            System.out.println("Done");

    }catch(IOException e){
        e.printStackTrace();
    }}
//////////////////////////////////////////////////////////////////////
after() : logdependency(){
    System.out.println("Log has been added!!");
```

No consoles to display at this time.

Fig.5. Advice invocation corresponding to exposed Join points



```
}
//////////////////////////////////////////////////////////////////////
after(String str): consoledependency(str)
{

try{
        //String data = " History of Actions \t";

        //System.out.print("in aspect block terminating nachos...");
        File file =new File("aspectfile.txt");
        if(!file.exists()){
            file.createNewFile();}
        FileWriter fileWritter = new FileWriter(file.getName(),true);
        BufferedWriter bufferWritter = new BufferedWriter(fileWritter);
        bufferWritter.write("\n");
        bufferWritter.write(str);
        bufferWritter.close();

}
catch(IOException e){
    e.printStackTrace();
}

}
//////////////////////////////////////////////////////////////////////
before() : logdependency() {
//System.out.print("Hello! ");
    try{
        String data = " Aspect Nachos BOOT at \t";
```

No consoles to display at this time.

Writable   Smart Insert   16 : 26

Fig.6. After advice invocation for the console handling

IV. RESULT ANALYSIS

A LOC Measurement

The lines of code were measured for both the aspect version of NachOS and the non-aspect version. The line count is done only for the features added to the existing machine code of the NachOS. The comparison is done on the basis of new functionalities introduced in the NachOS using aspect paradigm and object oriented paradigm. Here in the implementation no modification are required in the original file, only the aspect file needs to be modified in case of updating the existing functionality, adding new functionality, or removing existing functionality.

Advice part of the aspect is to be modified as per the requirements and the LOC is measured only in context of the original file

Table 1 below shows the LOC comparison of two implementations of various features added in the NachOS using two different programming paradigms. The aspect implementation does not require any modification in the original file of NachOS; LOC is only added in advice part of the aspect module. However if the same functionality is implemented without using aspects then significant LOC is to be added at every occurrence of the called function in the original file, which in turn affects the entire tangled code.

Table 1: LOC analysis after code introduced in the original file

New Functionality Added	**LOC added for new functionality using AOP**		**LOC added for new functionality without using AOP in Main file**
	LOC added In Machine Main File	**LOC added in aspect module**	
Adding Date and time before Logging event	0	8	8*No of occurrences of called function
Authentication	0	17	17* No of occurrences of called function
Console History	0	13	13* No of occurrences of called function

B Execution Time Analysis

The execution time analysis is shown in Figure 7 using the Eclipse Juno and AOP is implemented with AspectJ. Java programmers can divide the program code into various separate modules; each of the separated modules is actually a different aspect. These modules are merged at the join points present in the main core code by the AspectJ compiler into a single program.

The test was conducted separately for computing the execution time as per the both implementations that are OOP and AOP. As this experiment is carried out in multi-programming environment and the call of NachOS's process is not certain, the execution time was computed more than hundred times and gets the average of it as the join points are added to the NachOS. This is done only to obtain the correct and accurate execution time values. As shown in Fig.4.6, the graph of AOP and OOP is approximately merged over each other, which indicates that the execution time efficiency of AOP is not decreased although AOP need to capture the joint point and execute the advice. AOP achieves the equal execution efficiency as obtained using OOP.

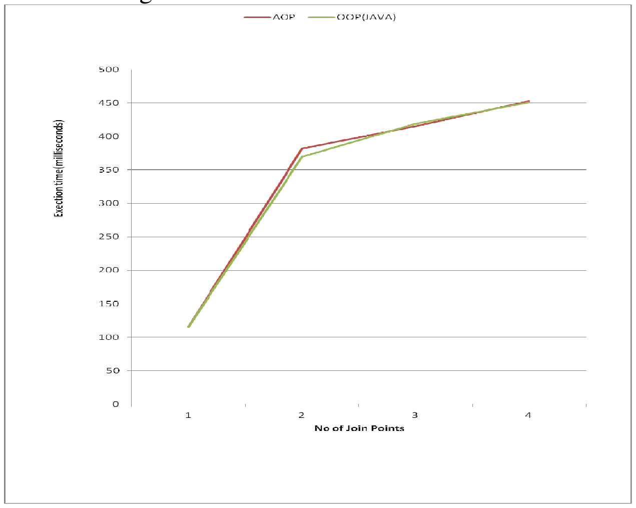


Fig.7. Execution time analysis

C Re-configurability

Due to the dynamic environment and frequently changing requirements, need arises to reconfigure a system. Reconfiguration cannot be done easily as the code which is to be modified is tangled and scattered throughout the system. Separation of concerned code is required which can be beneficial not only for re-configurability purpose but it could also enhances the extensibility, reusability and understand ability of the system. The reconfiguration is analyzed in two different implementations which are aspectized and non aspectized implementation. The non aspectized implementation is shown in the Table 4.3 for various concerns. The reconfiguration needs the modification at all the locations in the code where the concern is present in the Operating system code. This task is very cumbersome, as it consists of searching as well as then modifying the code for new configuration at all the locations where concern is present in the operating system's code.

Feature Reconfigured	Aspectized implementation	Non aspectized Implementation
Date and time log is to be modified	1	M
Authentication	1	N
Console Handling	1	O

Table 4.2 Modifications required at the various locations if reconfiguration arises

Where M,N,O are the number of locations where they are present in cross cutting fashion

Conclusion

*With th*e use of aspects in the OS, the tangled code is shifted from the main module into the aspect module thus making the design and implementation of OS clearer and understandable. Errors caused by the developers are now reduced as the developer does not need to write the redundant code or function's call to all the occurrences where it is to be executed. Rather than following this procedure, now the developer can easily invoke the function calls at all the places by using a single pointcut. Also the development process becomes faster and error free with the aspects, because the design is clearer and also the concerns are implemented separately from their tangled code.

The maintainability of Operating system is also enhanced in the AOP implemented code as the design consists of separated concerns and the required changes are applied only to the particular aspect that is associated with module to be maintained. Moreover the code is less scattered and tangled; the probability of progression of error in the system is also reduced.

References

[1] S. Hanenberg, S. Kleinschmager and M. Walter, "Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code", Third International Symposium on Empirical Software Engineering and Measurement, IEEE, Feb 2009

[2] Y. Zhang, et.all, "Implementing and Testing Producer-Consumer Problem using AOP", Fifth International Conference on Information Assurance and Security, IEEE, 2009

[3] Y.Coady, G. Kiczales, and M. Feeley, "Exploring an Aspect-Oriented Approach to Operating System Code", September 2000.

[4] A. Rashid, T. Cottenier, P. Greenwood and R. Chitchyan, "Aspect-Oriented Software Development in Practice", Computer Society IEEE, Feb 2010.

[5] G,C. Murphy, R, J. Walker, and E. L.A. Baniassad, "Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming". IEEE transactions on software engineering, vol. 25, no. 4, July 1999

[6] G. Murphy and C. Schwanninger, "Aspect-Oriented Programming", Computer Society, IEEE, 2006

[7] W. A. Christopher, S. J Procter and T. E. Anderson, "The NachOS Instructional Operating System", 2005.

[8] T. Narten, "A road map through NachOS", 1995

[9] J. Niu, "NachOS Overview", Operating Systems, CS-CCNY, Fall, September 2003

[10] S. Chiba and R. Ishikawa, "Aspect-Oriented Programming Beyond Dependency Injection", Springer-Verlag Berlin Heidelberg 2005

[11] J. Brichau, et all, "A Model Curriculum for Aspect-Oriented Software Development", IEEE Software, December 2006

[12] C. L. Anderson, and M, Nguyen, "A survey of contemporary Instructional Operating System for use in Undergraduate Courses". JCSC 21, Oct 2005

[13] R. Laddad," AspectJ in Action- Practical Aspect-Oriented Programming", Manning Publications co. 2003

[14] J.D. Gradecki, "Mastering AspectJ-Aspect Oriented Programming in Java", Wiley Publishing, Inc, 2003

[15] W. L. Lieu, C. H. .Lung, and S. Ajilla, "Impact of Aspect Oriented Programming on Software Performance: A Case Study of Leader/Followers and Half-Sync.

[16] J. Lamping and J. Kiczales, "The Need for Customizable Operating systems", IEEE, 1993

[17] M. Ceccato. and F. B. Kessler, "Migrating Object Oriented code to Aspect Oriented Programming", Software Maintenance, ICSM International Conference, IEEE, 2007

[18] http://www.ida.liu.se/~TDDB63/material/begguide/ (URL working on 2.4.2014)

[19] https://www.student.cs.uwaterloo.ca/~cs350/ common/osoverview.html (URL working on 2.3.2014)