ISSN No. 0976-5697

Volume 2, No. 6, Nov-Dec 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Deducing and Mapping the Class Path of JAR File

PM. Shanthi*
Associate Professor
Department of MCA
K.S.Rangasamy College of Technology
Tiruchengode ,India
shanthimuthumani@yahoomail.com

Dr. K. Duraiswamy
Professor
Department of MCA
K.S.Rangasamy College of Technology
Tiruchengode, India
drkduraiswamy@yahoo.co.in

Abstract: Now-a-days, to reduce the effort of human and to improve the efficiency of the process, everything in this world is being computerized. But also the user may face some problems while executing the program. Program is software that contains a set of instructions to perform a task. In order to make the process of developing program, an easier one, some developers develop software and implement them as a package.

To access these packages much easier, some of them are released as an Open Source Software. An Open Source Software is a software that is downloaded with free of cost. In this Open Source Software, there exist many packages, classes and methods. While using this Open Source Software, to develop an application, these packages are well-connected with that application. It is necessary to generate a class file to start executing the application. But the user doesn't know the classes or its properties to compile them.

To overcome this situation, in this paper, we propose a method to start compiling the classes in the package automatically. Also if the path is not set correctly, it is also done systematically through the methodology proposed in this paper.

Keywords: Classes, Compilation, Computerized, Developers, Download, Efficiency, Instructions, Methods, Open Source Software, Package, Program, Systematically.

I. INTRODUCTION

Computer and Internet are the two essential things in this world, to lead a life easier. Almost all the things in this world are being computerized, to make it fast and efficient. To do this, everything is organized as a program. Program consists of instructions which instruct the system about the process to be carried out. To develop these programs, some of these instructions may be repeated for a certain number of times, such as for either input or output. In this case, these instructions are grouped to form packages. These packages can have classes, methods, variables and so on.

There exist some pre-defined packages along with the software, in which the program is to be developed. Since these packages are in-built with the software, we can able to use it while doing the program in that software. Some software is released as an Open Source Software to make more number of users accessing the software.

Open Source Software is the software that can be some specific features and it can be downloaded with free of cost. These open source software occupies less memory space, since it can be downloaded and used by all kinds of users and to enable more number of users to use it.

This Open Source Software may also contain pre-defined packages, JAR files and so on to make the program to be developed in an efficient manner. While choosing the open source software, it may be possible to use all the packages in the software. But the user may not know about the number of classes in the packages and also about the number of packages or JAR files.

To make this possible, as defined in the paper [1], an automatic search engine is implemented to count the number of packages in the software, number of classes in the packages and so on. Also, as defined in the paper [2], an algorithm is proposed to compile the packages automatically and to search for the error occurrence. From the error, the error rate is calculated automatically. And from the error rate, the value of the software is concluded.

In some situation, there may be the necessary to use the JAR files in the software. JAR files are the files that are defined as the Java ARchive Files, which are the already compiled file. The JAR File is developed by the programmer and it is to be converted into a compiled file. This compiled file is then become available for use. This compiled file is of error-free. To use this compiled file, the class path of the JAR is given in a proper manner. A Proper setup of the class path makes the JAR file and its properties to be implemented in the program. The JAR files can also be implemented and use like a package.

If the class path setup becomes wrong, the JAR file does not able to map with the program and so the program does not able to retrieve the functionality of the JAR files. In that case, it is necessary to correct the class path setup. But this JAR file is a pre-defined one, the user cannot able to identify the path of the file and so they might not able to correct the wrong path.

In that situation, they might be in need of some help. To help the user in this situation, this paper proposes an efficient methodology, to detect and correct the path automatically. This is due to the user cannot know about the source file for the JAR programs. So the search engine automatically searches for the class files in the package, and then it set up the class path to proper implementation of the methodology into the program.

The summary of the process that has to be automated to make the process efficient are as follows:

a. Open the software and analyze it.
b. Identify the number of packages, classes and methods in that software.
c. Identify the name of these packages, classes and methods.
d. Identify the JAR files in the software.
e. Find the class file for the JAR.

f. Set up the path for the JAR file.

Thus in this paper, an efficient methodology is to be implemented in our proposed work to make all the above mentioned process automated. Thus this paper solves all the problems faced by the users while using the open source software.

The organization of the paper is as follows: This paper contains the related papers that are used to develop this research work in section II. In section III, the proposed methodology is discussed with proper algorithm and flowchart. In section IV, the experimental setup is described which will explain about the real time implementation of the methodology proposed in this paper.

II. RELATED WORK

To implement the methodology in our proposed work, the papers we referred are given below:

As computers become an integral part of today's society, making them dependable becomes increasingly important. Field studies [3] and everyday experience make it clear that the dominant cause of failures today is software faults, both in the application and system layers. Reducing the number of software faults and surviving the ones that remain is therefore an important challenge for the fault-tolerance community.

Open source programs share a number of important characteristics. First, they are widely used and hence form a relevant base of software to study. For example, the Apache web server is used by 54% of all web sites [4]. Second, their development process is open.

In particular, we wish to test the hypothesis that generic (i.e. not application-specific) recovery techniques, such as process pairs, can survive a majority of application faults. Our methodology differs from that of prior studies [5]. We reason from bug reports and source code as to whether a purely generic recovery system would have recovered from application faults, while past studies examine the field behavior of implemented, mostly generic recovery systems.

Faults may be classified into two categories: operational and design [3]. Operational faults are caused by conditions such as wear-out and can be handled with simple replication. Faults caused by design bugs are much more difficult to handle, because simply replicating a buggy design often results in dependent failures in which all the replicas fail. Software faults are difficult to survive because they are all caused by design bugs.

Nair et al. [6] described a case study of combinatorial testing for a small subsystem of a screen-based administrative database. The system was designed to present users with input screens, accept data, then process it and store it in a database. Size was not given, but similar systems normally range from a few hundred to a few thousand lines of code. This study was extremely limited in that only one screen of a subsystem with two known faults was involved, but pair wise testing was sufficient to detect both faults.

Wallace and Kuhn [7] reviewed 15 years of medical device recall data gathered by the US Food and Drug Administration (FDA) to characterize the types of faults that occur in this application domain. These applications include any devices under FDA authority, but are primarily small to medium sized embedded systems, and would range from roughly 104 to 105 lines of code. All of the applications in the database were fielded systems that had been recalled because of reported defects. A limitation of this study, however, was that only 109 of the 342 recalls of software-controlled devices contained enough information to determine the number of conditions required to replicate a given failure. Of these 109 cases, 97 percent of the reported flaws could be detected by testing all pairs of parameter settings, and only three of the recalls had an FTFI number greater than 2. (The number of failures triggered by a single condition was not given in [7], but we reviewed the data.) The most complex of these failures required four conditions. Kuhn and Reilly [8] analyzed reports in bug tracking databases for open source browser and server software, the Mozilla web browser and Apache server. Both were early releases that were undergoing incremental development. This study found that more than 70 percent of documented failures were triggered by only one or two conditions, and that no failure had an FTFI number greater than 6. Difficulty in interpreting some of the failure reports led to conservative assumptions regarding failure causes.

Thus, some of the failures with high FTFI numbers may actually have been less than 6. Three other studies provided some limited information regarding fault interactions. Dalal et al. [9] demonstrated the effectiveness of pair wise testing in four case studies but did not investigate higher-degree interactions. Smith et al. [10] investigated pair wise testing of the Remote Agent Experiment (RAX) software on NASA's Deep Space 1 mission. The RAX is an expert system that generates plans to carry out spacecraft operations without human intervention. This study found that testing all pairs of input values detected over 80 percent of the bugs classified as either "correctness" or "convergence" flaws in onboard planning software (i.e., successfully finding a feasible path), but only about half of engine and interface bugs [10]. The authors did not investigate higher-degree combinations required to trigger a failure. Pan [11] found that testing all values triggered more than 80 percent of detected errors in a selection of POSIX operating system function calls from 15 fielded commercial systems. Higher-degree combinations were not reported.

Network layer transmission errors—which have been considered highly improbable for moderate-sized clusters—cannot be ignored when dealing with large scale computations [12]. Additionally, the probability that a parallel application will encounter a process failure during its run increases with the number of processors that it uses. If the application is to survive a process failure without having to restart from the beginning, it either must regularly write checkpoint files (and restart the application from the last consistent checkpoint [13,14]) or the application itself must be able to adaptively handle process failures during runtime [15]. All of these issues are current, relevant research topics. Indeed, some have been addressed at various levels by different projects. However, no MPI implementation is currently capable of addressing all of them comprehensively. This directly implies that a new MPI implementation is necessary: one that is capable of providing a framework to address important issues in emerging networks and architectures. Building upon prior research, and influenced by experience gained from the code bases of the LAM/MPI [16], LA-MPI [12], and FT-MPI [15] projects, Open MPI is an all-new, production-quality MPI-2 implementation.

In the paper [15] they describe a framework called Columbus [17] with which we are able to calculate the object oriented metrics validated in [18], [19], [20] and [21] for fault-proneness detection (similar results were presented also in [22] and [23]) from the source code of the well-known open source web and e-mail suite called Mozilla [24, 25]. We then compare our results with those presented in [18].

Based upon these surveys, this paper is proposed with much better simulation model.

III. METHODOLOGY

A. Proposed Method:

The aim of the proposed work is to propose a methodology to compile the package automatically and to set up the proper class path for implementing the methods in JAR files.

The summary of the proposed work is as follows: First, the user selects the software to develop their program. Upon selecting the software, the methodology proposed in this paper has to compile the software to identify the errors. From the deduced errors, the error rate is to be found out.

After calculating the error rate, our work has to go through the errors and identify the reporting statement about the error. The error may occur due to the pre-defined packages or due to the JAR files. If the pre-defined packages contain error, it may be calculated as error rate. If the JAR file connected with the program contains error, it may be due to wrong mapping of the class file of JAR or due to not set up the path properly. If this is the case, then the methodology of this paper proceeds as follows:

The user might not able to identify the class files in the JAR, since it is already a compiled program. The user cant able to know about the properties and execution of the JAR files. In this paper, the automatic search engine has to search for the class files in the JAR and the class path is set up properly. This automatic process helps the user to run the software without any delay.

And the user may be free from knowing about the JAR files and the class path set up.

The process of locating the JAR files is carried out as follows:

a. Identify the Errors.
b. If the error is due to incorrect mapping of the JAR file, then the JAR file is analyzed to identify the class path.
c. Detect the class path and map it with the program.
d. Execute it after setting up of class path. Thus the JAR file is implemented correctly.

B. Algorithm:

The proposed method proposes an algorithm to make the process of compiling and mapping the JAR files automatically with the program. The algorithm is given below:

```
Start
Compile the program automatically
Detect the errors and calculate the error rate.
Identify the errors and categorize it (Either due to package error or due to JAR file mapping error)
If error due to package error
    Add it with the error rate
Else if error due to the mapping of JAR file
    Analyze the JAR file
    Find the Class path
    Set up the correct class path
    Execute the program
End if
Stop
```

IV. EXPERIMENTAL RESULTS

The proposed work has to undergone for an experiment to test for its performance. The experimental setup and the result are discussed in this section.

The experimental setup is undertaken by taking a set of programmers. The programmers are instructed to develop a program by using the open source software. And their program must inherit the properties and classes of the packages in the software. Also the program must inherit the JAR files in its development.

Upon developing the program, the program is compiled by the automation search engine proposed in the paper [2] and deduces the compilation errors. From the compilation results, the error rate is deduced and then the errors are categorized. From the classification, if it is found that the error has been occurred due to the class path of the JAR file, the algorithm is used to detect the correct class path. And the execution is made automatically upon setting up of the class path.

Thus the proposed method is implemented with proper experimental setup and the mapping is done automatically.

V. COMPARISON CHART

To explain the efficiency of our work, we took some open source software for our analysis. The performance of the open source software before the implementation of our algorithm and the performance of that software after the implementation of our algorithm is shown below: Also the comparison chart with suitable data is given.

Table -1: Execution Rate (Before Implementation)

Software	Execution Rate (in %)
ActiveMQ	50
Apache	78
BeanShell	95
BlueJ	25
IBM	24
Eclipse	68
EJB Benchmark	87
Hibernate	90
Jasper	91

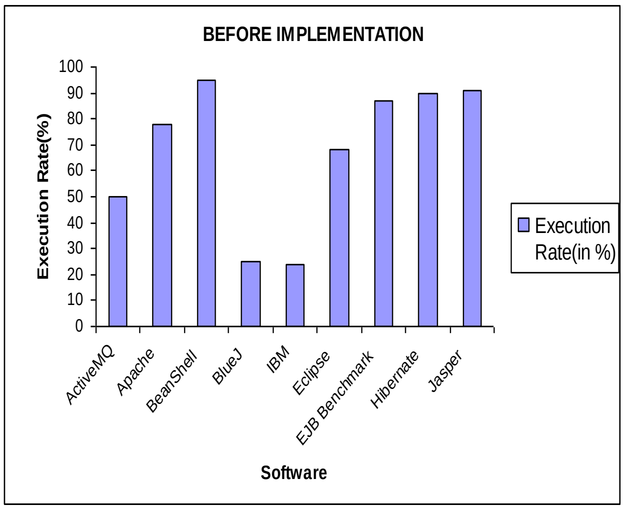

Chart -1: Execution Rate (Before Implementation)

Table -2: Execution Rate (After Implementation)

Software	Execution Rate (in %)
ActiveMQ	80
Apache	82
BeanShell	84
BlueJ	75
IBM	75
Eclipse	80
EJB Benchmark	90
Hibernate	95
Jasper	95

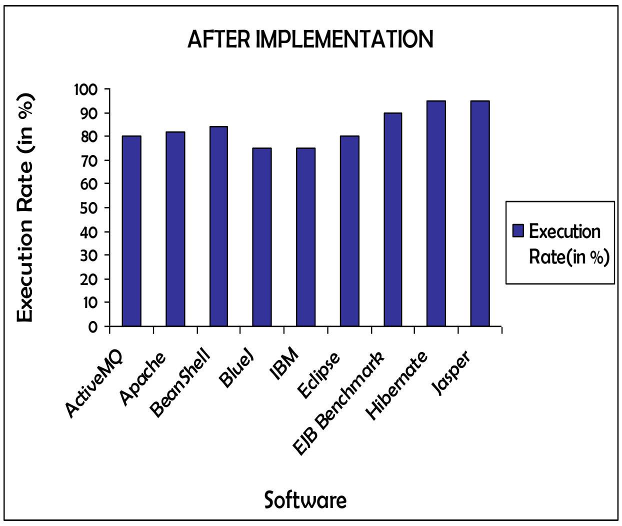

Chart -2: Execution Rate (After Implementation)

Thus the comparison is made as shown in the table-1 and table-2 and the chart is shown in the chart-1 and chart-2.

VI. CONCLUSION

The aim of the paper is to develop an automation search engine to deduce the error occur in the JAR file mapping and correct the mapping automatically by detecting the correct path of the JAR file.

This paper proposes an efficient methodology to achieve the aim of the paper. This is useful in the development of the program with the use of JAR files. JAR files are the files that are error-free. If the proper implementation of the class path to the JAR files, the program has to be developed successfully.

Thus this paper contains efficient methodology to reduce the error occurrence due to the JAR file mapping.

VII. REFERENCES

[1] “An Empirical Validation of Software Quality Metric Suites on Open Source Software for Fault-Proneness Prediction in Object Oriented Systems”, 2011 publications.

[2] “An Adaptive Fault Tolerance in Open Source Software”

[3] Jim Gray and Daniel P. Siewiorek. High-Availability, Computer Systems. IEEE Computer, 24(9):39–48, September 1991.

[4] The Netcraft Web Server Survey. At http:// www.netcraft.com/survey/

[5] I. Lee and R. Iyer. Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARD IAN Operating System. In International Symposium on Fault-Tolerant Computing (FTCS), pages 20–29, 1993.

[6] V.N. Nair, D.A. James, W.K. Erlich, and J. Zevallos, “A Statistical Assessment of Some Software Testing Strategies and Application of Experimental Design Techniques,” Statistica Sinica, vol. 8, no. 1, pp. 165- 184, 1998.

[7] D.R. Wallace and D.R. Kuhn, “Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data,” Int’l J. Reliability, Quality and Safety Eng., vol. 8, no. 4, 2001.

[8] D.R. Kuhn and M.J. Reilly, “An Investigation of the Applicability of Design of Experiments to Software Testing,” Proc. 27th NASA/IEEE Software Eng. Workshop, Dec. 2002.

[9] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz, “Model-Based Testing in Practice,” Proc. Int’l Conf. Software Eng., 1999.

[10] B. Smith, M.S. Feather, and N. Muscettola, “Challenges and Methods in Testing the Remote Agent Planner,” Proc. Fifth Int’l Conf. Artificial Intelligence Planning Systems, 2000.

[11] J. Pan, “The Dimensionality of Failures—A Fault Model for Characterizing Software Robustness,” Proc. Int’l Symp. Fault-Tolerant Computing, June 1999.

[12] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant messagepassing system for terascale clusters. International Journal of Parallel Programming, 31(4):285–303, August 2003.

[13] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In SC’2002 Conference CD, Baltimore, MD, 2002. IEEE/ACM SIGARCH. pap298,LRI.

[14] Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, and Eric Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. International Journal of High Performance Computing Applications, To appear, 2004.

[15] G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovski, and J. J. Dongarra. Fault tolerant communication library and applications for high perofrmance. In Los Alamos Computer Science Institute Symposium, Santa Fee, NM, October 27-29 2003.

[16] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In Proceedings, 10th European PVM/MPI Users’ Group Meeting, number 2840 in Lecture Notes in Computer Science, Venice, Italy, Sept. 2003. Springer.

[17] R. Ferenc, A´ . Besze´des, M. Tarkiainen, and T. Gyimo´thy. Columbus – Reverse Engineering Tool and Schema for C++. In Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002), pages 172–181. IEEE Computer Society, Oct. 2002.

[18] V. R. Basili, L. C. Briand, and W. L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. In IEEE Transactions on Software Engineering, volume 22, pages 751–761, Oct. 1996.

[19] L. C. Briand, W. L. Melo, and J. W¨ust. Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects. In IEEE Transactions on Software Engineering, volume 28, pages 706–720, 2002.

[20] L. C. Briand and J.W¨ust. Empirical Studies of Quality Models in Object-Oriented Systems. In Advances in Computers, volume 56, Sept. 2002.

[21] L. C. Briand, J. W¨ust, J. W. Daly, and D. V. Porter. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. In The Journal of Systems and Software, volume 51, pages 245–273, 2000.

[22] F. Fioravanti and P. Nesi. A Study on Fault-Proneness Detection of Object-Oriented Systems. In Fifth European

Conference on Software Maintenance and Reengineering (CSMR 2001), pages 121–130, 2001.

[23] P. Yu, T. Syst¨a, and H. M¨uller. Predicting Fault-Proneness using OO Metrics: An Industrial Case Study. In Sixth European Conference on Software Maintenance and Reengineering (CSMR 2002), pages 99–107, 2002.

[24] The Mozilla Homepage. http://www.mozilla.org.

[25] C. R. Reis and R. P. de Mattos Fortes. An Overview of the Software Engineering Process and Tools in the Mozilla Project. In Proceedings of the Workshop on Open Source Software Development, pages 155–175, Feb. 2002.