International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org Volume 2 Issue 7 ǁ July. 2013 ǁ PP.29-35

Adopting Aspect Oriented Programming in Enterprise Wide Computing

Dr. Debashis Jena [1], Dr. S.C. Das [2,] Ambika Prasad Das [3]
[1](*Department of Computer Science and Engineering / IIIT Bhubaneswar, India)*
[2](*Department of IT Management, KIIT School of Management, Bhubaneswar, India)*
[3](*Department of IT Management, KIIT School of Management, Bhubaneswar, India)*

ABSTRACT: *Aspect Oriented Programming (AOP) has received the attention of both the research community and industry experts since its inception in Xerox Palo Alto Research Center (PARC) in 1996. This supplements the Object Oriented Programming paradigm by separating the cross cutting concerns from business logic. MIT in its technical review in 2001 identified AOP as one of the ten emerging technologies that will have significant impact on the economy. However even after more than a decade of its inception as a programming paradigm, adoption of AOP in the IT industry in general and IT services industry in particular is still in its infancy. This paper discusses some of the pain areas in adoption of AOP in the IT industry.*

KEYWORDS: *adoption, Aspect Oriented Programming (AOP), cross cutting concerns, Object Oriented Programming (OOP).*

I. INTRODUCTION

The In today's complex business scenario managing IT projects has several challenges. IT managers are faced with risk arising out of single / preferred vendor, conflicting standards, designs, maintainability issues and high cost due to use of proprietary standards and tools. Therefore more and more non IT companies follow collaborating model while outsourcing IT products and services which can cater to their organization's needs. While collaborative software development reduces single vendor dependency, it also adds complexity due to conflicting standards, rework due to defect in collaborating partner's modules and integration challenges.

Object Oriented Programming (OOP) model revolutionized software development with the facility of modeling real life objects in programming languages thereby reducing complexity and increasing modularity. However as the size and complexity of software applications grew, modeling behavior of software components and their modularization became increasingly important. Also separation between application programmers and system programmers became more and more discrete. Orthogonal concerns like transaction, security, logging and exception handling needed to be implemented in a distributed, heterogeneous environment thereby separating enterprise wide system concerns from programming logic. Also these concerns or functionalities needed to be more and more robust which can't be implemented within limited timelines of the project.

In collaborative software development many outsourcing partners get involved based on their domain competency, skill set, team size and other pressing business requirements. IT organizations have tough time managing these diverse teams and ensuring project completion within time and budget. While there can be many contributing factor for project overrun, integration is one of the major issues in such projects.

Aspect Oriented Programming (AOP) helps in addressing cross cutting concerns which in turn is supposed to reduce cost and complexity. The objective of this research is to identify pain areas in large scale software development using enterprise wide computing standards and critically appraise scenarios where AOP can do value addition in terms of reducing cost, integration challenges and maintenance overheads.

II. HISTORY AND EVOLUTION OF AOP RESEARCH

Gregor Kiczales and colleagues at Xerox Palo Alto Research Center (PARC) developed the explicit concept of AOP sometime between November 1995 and May 1996. AOP was based on a strong foundation of prior work, but somehow the existing terminology was not sufficient to describe what was being done. AOP revolutionized the way programming was being done and very soon became an interest of researchers around the world. [2]

The RG Case Study

One of the early projects at PARC which contributed to the development of AOP as a paradigm was RG (Reverse Graphics). RG is an image processing system that allows sophisticated image processing operations to be defined by composing primitive image processing filters. An implementation of RG using OOP was easy to implement but has performance bottlenecks. These performance issues could not be addressed effectively in a

paradigm tied to the object boundaries, rather required a mechanism in which these complexities cross cutting several modules to be handled as separate concerns. The paper on RG case study [3] discusses how this limitation of OOP was overcome using Aspect Oriented Programming techniques. The RG case study demonstrated that AOP can reduce code complexity without sacrificing important performance requirements.

Annotated MatLab (AML)

The second project that contributed to the development of AOP paradigm during this period was Annotated MatLab (AML) [5]. The problem discussed here was optimization of certain MatLab programs, again focusing on memory usage and operation fusion. There were many discussions among the group at PARC whether AML was AOP or not. The final language annotations did not match with other AOP systems at PARC due to which AML did not get much importance in AOP research. However it served to list out features which should not be part of AOP.

Evaluation Time Control Meta Language (ECTML)

This work was carried out by John lamping. He conceptualized a small system called Evaluation Time Control Meta Language (ECTML) where idea was to provide a set of directives that programmers could use in order to instruct the language processor when to evaluate certain parts of the code. This work was analogous to work on reflection where processor decides certain parts of the program to be evaluated at compile time or runtime. The gist of this work was that language processors were not always capable to determine the best time to evaluate these parts of the code and explicit specification from the programmer would help the processors in this aspect. This contributed to another dimension of separation of concerns enriching AOP knowledgebase.

DJ

DJ was different in its approach compared to RG, AML and ETCML. DJ was a Java package for adaptive programming. Adaptive programming enables the programmer to practice concern-shy programming. DJ made it easier to follow the Low of Demeter, loosening coupling between the structure and behavior concerns and adapting changes in the object model. [4].

AspectJ

AspectJ [6] was another milestone project in Aspect-Oriented research by PARC and was first made public in 1998. This is a simple and practical aspect oriented extension to Java language. Effort was being made to make AspectJ a general purpose AOP implementation whereas its predecessors like DJ and RG were concern and domain specific in nature respectively. AspectJ was designed to be a compatible extension to Java so that it will facilitate adoption by existing Java programmers' community.

III. RELATED WORK

Since its inception, AOP researchers have been interested in the benefits which AOP brings to the table in terms of the software quality parameters. However there is a significant research gap when it comes to the benefits and challenges of managing enterprise wide computing and enterprise architecture by adopting AOP in place of plain OOP methodologies in a multi vendor environment. Also there seems to be little ongoing research on the costs and effects of AOP within the enterprise.

Kiczales etal [1] in the proceedings of ECOOP, 1997 identified the programming problems with OOP where OOP alone fails to capture all important design decisions that the program must implement. In fact there are some programming problems that fit neither the OOP nor the procedural approach it replaces. The paper presents an example driven approach of the relevance of AOP in software architecture. First example is the image processing example done in RG and AOP based reimplementation of the same. They found that the AOP based reimplementation of the system has met the original design goals, application code is easy to interpret and maintain in addition to being highly efficient. They have agreed that it is difficult to measure the benefits of using AOP without a large experimental study involving multiple developers. They measured the effectiveness of AOP by taking into consideration the tangled code (code without AOP) size, component program size and sum of aspect program size. In this experiment they have found extremely large gain so far code size and complexity was concerned. However there were no clues on the integration efforts and complexity arising out of using a full-fledged AOP implementation and the costs involved.

In the second example, they implemented a distributed digital library that stores documents in many forms and provides a wide range of operations on those documents. There they studied various aspects like communication where a higher aspect level language was found to be appropriate vis-à-vis traditional reflective access to control communication. One of the major goals which were still an open issue in this paper was quantitative assessment of the utility of AOP. How much AOP helps in maintenance? Which applications /domains / industries it is more applicable for? These were still open questions.

Roger T. Alexander and James M. Bieman [8] in their paper 'Challenges of Aspect-oriented Technology' discussed on some of the pain areas in implementing AOP. The paper focuses on certain aspects

like understandability, fault tolerance, cognitive burden etc and stresses that there are some of the grey areas when it come to effective implementation of AOP in practical situation. For example in case of fault tolerance diagnosing the root cause will involve examining not only the primary abstraction but also the woven aspects. This increases the overhead of using AOP effectively with realizable benefits. However the complexity arising out of weaving process was still not measurable with accuracy. Similarly maintainability issues could not be addressed. This is because changes to the primary abstraction that forms the basis of a woven composition have potential to require changes to the woven aspects.

Avadhesh Kumar, Rajesh Kumar and P.S. Grover studied the maintainability of Aspect Oriented Systems [10] and have found that average change impact in AO systems is less than that of OO systems which suggests maintainability is better in case of AO systems. However they concluded that during reengineering if concerns which are not cross cutting in nature are mined to aspects, this will make the system more complex and in turn less maintainable.

Zhao [9] did some work in this area of change impact analysis based on program slicing technique. However, this was not applied this to realistic systems.

Joon-Sang Leea and Doo-Hwan Baeb in 2002 proposed an Aspect Oriented Development Framework (AODF) [11] in which functional behaviors are encapsulated in each component and connector and particular non functional requirements are tuned flexibly during software composition. AODF would enable intra component behavior with the component and inter component behavior with the connector using a CB style. In order to illustrate how well this framework works, they have used OpenJava providing reflective support during compile time. This work was not complete, but provided some light on feature component writing standard and architectural style to implement intra-component, inter-component behavior and non functional behavior in component based software developments.

Roberta Coelho et al in 2010 [12] identified some drawbacks of aspect oriented programs when it comes to exception handling. In this work they presented a verification approach based on a static analysis tool called SAFE, to check the reliability of exception handling code in AspectJ programs. They stressed on increased complexity arising out of additional code included by the aspect weavers and their resolution by application programmers. Secondly most AOP implementations work on the basis of inversion of control where the aspect has the control which classes it should affect rather than the class itself deciding so. This means the advised code can't be safe from the exceptions flowing out of the aspects which are challenging to debug. Also AO paradigm mandates application and aspect code being developed in parallel. This leads to obliviousness on part of the application programmer about the aspect code and exceptions flowing out of aspects into the base code are not handled with ease by the application developer.

Jianjun Zhao studied the opportunities and challenges of AOP software maintenance [13]. He reiterated that AOP research has primarily been focused on problem analysis, language design and implementation. Even though issues related to software maintenance are known, it has received little attention. AOP programs consist of base code and aspect code which is weaved into the base code to address cross cutting concern. Weaving is what makes it different from OO or procedural programs. Therefore in order to maintain AO software effectively, new analysis and testing techniques are strongly needed.

Gail C. Murphy et al studied the lessons learned from accessing AOP [14]. They found that while evaluating a software engineering methodology, the researcher must be aware of 3 factors which require trade off with each other. They are – validity, realism and cost. To study AOP, they applied two basic methods – a case study method and an experimental method. Since the technique under study is in its infancy, the case study and experimental methods were largely exploratory, yielding qualitative insights into AOP and directions for further investigation. Overall they have found the case study approach more useful in evaluating how AOP helps ease some of the tasks of software developers.

Cristina Videira Lopes and Martin Lippert in their paper 'A Study on Exception Detection and Handling Using Aspect-Oriented Programming' [15] studied the exception handling capabilities of AOP. Their work was mainly focused on accessing APO's capabilities in easing out code tangling in handling exceptions. They have found that AspectJ helps reducing code related to exception detection and handling to a substantial extent. Even in one of the scenarios they were able to reduce the code related to exception handling by a factor of 4. Also they have found that in contrast to plain Java code, AspectJ provided better configuration of different exceptional behavior and more exception tolerance in case of change in specification. Their study also identified some drawbacks of AspectJ which can be addressed in future.

Freddy Munoz et al in the year 2009 did some empirical study on the usage of AOP [16]. In 2001 MIT had announced AOP as a key technology that will soon have profound impact on the economy and on how we live and work [17]. However, the usage of AOP is not widely adopted even today as it was predicted. This can be attributed to lack of matured tools for analysis, maintenance and testing. In this paper they analyzed the usage of AOP in 38 open source aspect-oriented projects from small (<5000 LOC) to large (>20000 LOC). Their objective was to find out to what extent developers use AOP, its invasiveness facilities and the coverage of

AspectJ Point Cut Descriptor (PCD) language. They observed that developers use few advices to modularize cross cutting concerns and that advices are scarcely cross cutting. There were other observations where it was found that developers write very few advices which break object-oriented encapsulation and the small number of invasive advice advise a small number of specific join points.
They have listed some possible reasons which can be interesting to the research problem

- Developer lack comprehensive knowledge of AOP and use of advice in modularizing concerns.
- Developers fail to appreciate units which seem modular but cross cut other modules.
- AspectJ implementation is not flexible enough to allow developers modularizing total of cross cutting concerns.
- The invasive capabilities of AspectJ which should be used modularizing cross cutting concerns are not used because they can introduce side effects.

The points concluded give an indication regarding the challenges in adopting AOP within an enterprise. Uirá Kulesza et al did some study on the effects of AOP with respect to maintenance aspect of the software [18]. The study involves a systematic comparison between object and aspect oriented versions of the same application in order to access to what extent each solution provides maintainable software decompositions. The analysis is base on fundamental attributes for modularity such as coupling, cohesion, conciseness and separation of concerns. They have found that AOP exhibits superior stability and reusability through the changes since it results in fewer lines of code, improved separation of concerns, more loose coupling and lower intra component complexity.

Jörg Kienzle et al in their paper on concurrency and failure of AOP [19] pointed out two important facts regarding the behavior of AOP. Firstly AO languages like any other macro language can be beneficial for code factorization. However code factorization using AOP should be done by experienced programmers.
Secondly concurrency and failures are specific concerns which are hard to aspectize unlike other concerns like logging and exception handling. As per their opinion, these are part of the phenomenon that objects should simulate. They used AspectJ and OPTIMA transactional framework to conduct their research. They observed the following so far transactions using AOP is concerned

- Aspectizing transactions by automatically applying transactions to previously non transactional code is doomed to fail. This is because of the transaction serialization and incompatibility of the methods provided by shared objects.
- Separation of transactional interfaces from rest of the programs is possible using AOP. However, this separation may be redundant where transactional aspect is actually part of the object it applies to. This can result in confusing code.
- AOP provides mechanism whereby application programmer can specify the transactional attributes as in case of EJB. However these are prone to error where an experienced programmer can break the ACID properties by specifying a wrong attribute.

IV. USE OF AOP IN ENTERPRISE

As per the MIT technological review in January / February 2001, ten emerging areas of technology will soon have profound impact on economy and how we live and work. [17] One of these top ten emerging technologies was AOP. However even today, the use of AOP is not as widespread as it was supposed to be. This is an open area of research which needs to be investigated.

What is an enterprise?

The Open Group Architecture Framework (TOGAF) defines “enterprise” as any collection of organizations that has a common set of goals. For example, an enterprise could be a government agency, a whole corporation, a division of a corporation, a single department, or a chain of geographically distant organizations linked together by common ownership [20].The common set of goals within the enterprise mandates the enterprise architecture to optimize across the enterprise the fragmented legacy of processes into an integrated environment and this is where AOP fits in. For example, it might be required to record failed financial transactions across all divisions into a single integrated repository for reporting and analysis. Such requirements require enterprise architecture to be flexible enough to handle these concerns in a unified, seamless way independent of the functional requirements.

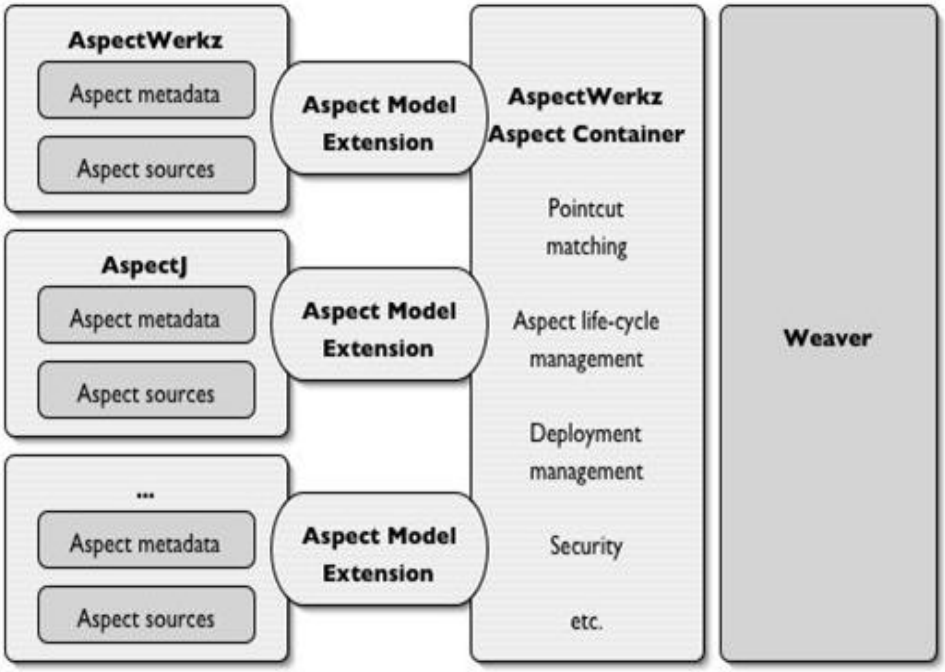
Enterprise Architecture

As per open group, the term "enterprise" in the context of "enterprise architecture" can be used to denote both an entire enterprise - encompassing all of its information and technology services, processes, and infrastructure - and a specific domain within the enterprise. In both cases, the architecture crosses multiple systems, and multiple functional groups within the enterprise. [20] Since architecture must cross cut multiple

systems and functional groups, separating the concerns from business functionality is of immense importance to the foundation of any enterprise architecture. AOP addresses this need by providing a way to address orthogonal concerns in a unified approach.

Using AOP in Enterprise Architecture

Separation of concerns is the biggest advantage AOP brings to the table apart from reducing code tangling, coupling and helping in architecture enforcement. Paulo Merson in his work 'Using Aspect-Oriented Programming to Enforce Architecture' [21] has stressed the importance of architecture enforcement using AOP. He demonstrated how coding policies, best practices and even naming conventions can be enforced using AOP. Architects may be having good confidence on the architectural artifacts which they produce, but may not have enough checks and means to ensure they will be adhered to in entirety. While architectural constraints can be enforced using AOP, its cost in terms of system processing and development time can be significant. However some basic architectural principles can definitely be imposed using AOP. AOP also adds to the quality of the architecture by ensuring modularity, maintainability etc.



Reference: http://www.theserverside.com/news/1365141/AspectWerkz-20-An-Extensible-Aspect-Container

Industry Adoption of AOP

There are many successful case studies of adoption of AOP in the enterprise. Motorola successfully developed WEAVR, an aspect oriented Modeling engine in light of the particular needs of telecom systems engineering industry. [25] One of the core business units of Motorola, the Networks and Enterprise business unit has successfully adopted AOP and is using WEAVR framework in production by development teams within the business unit. It is interesting to note that right now, adoption is limited to simple aspects like tracing and time out concerns. The maturity of WEAVR is still in its infancy.

HP successfully adopted AOP in C++ frameworks used in the development of VLSI CAD applications. [27] The result shows reduction of code and improved modularity by adopting AOP.

Siemens adopted AOP by developing eclipse based plug-in and through that access to aspect repository. [26] Siemens team demonstrated benefits of AOP by using an ordinary Java application and aspectizing it for better architectural benefits.

SAP is the world's largest enterprise software vendor. Therefore SAP research on AOP is very much relevant in the area of adopting AOP in large enterprise systems. Christoph Pohl et al have analyzed how AOP would help SAP to tackle hard development problems and which roadblock could prevent its adoption. [24] They have analyzed Enhancement Framework developed using ABAP to access the adoption issues related to AOP. They observed that AOP has been used in a couple of industrial projects in SAP. However, the adoption of AOP was not up to the expectation in large scale enterprise applications. Reasons are often a combination of social, technical, psychological and commercial considerations that are worth investigating. They have identified the following factors which inhibit AOP adoption in enterprise computing.

Social and Psychological Factors – Survey in internal SAP newsgroups indicated lack of awareness of AOP. Therefore AOP is not considered as an option in many enterprise software projects.
Optimal adoption of AOP requires experienced professionals which make the situation worse. Since ABAP does not mandate object oriented programming, applying AOP on ABAP codebase requires senior personnel which is not always justifiable by the organization.
Developers often feel losing control when AOP is introduced, because the aspect code might affect their base code and they might be responsible for errors they can't influence.

Technical Factors – For understanding flow at runtime involving various aspects, powerful tool support for tracing, verifying, debugging is of utmost importance. While some implementations are available for Java, tool support in ABAP is not very strong. Secondly, the risks arising out of Aspect based coupling and code interactions become more when third party software vendors are involved. Strict governance models are required to avoid side effects.

Economic Factors – It is really difficult to calculate the difference in ROI between projects with or without AOP. The ROI calculation of AOP platform extensions like Enhancement Framework is even more difficult.
However these are case studies only from the IT products sector. The penetration of AOP in the services sector is still scarce and this is an open research question. Some of the contributing factors to low penetration of AOP within enterprise are listed below.
Pain areas in Adopting AOP within the enterprise
There can be many contributing factors which prevent adoption of AOP optimally within the enterprise. Some of the reasons are listed below.

- ***Awareness:*** Though AOP was identified as one of the technologies by MIT in 2001 that would impact economy as a whole in the next decade, it never got the kind of proliferation and importance among the researchers and industry leaders its predecessors like OOP had. Firstly AOP emerged from an organization which specializes in image processing. Therefore perhaps the perception it got for quite some time was that it was specific only to image processing domain. Many programmers still seem to be oblivious of AOP as a programming paradigm.
- ***Lack of common AOP framework:*** The major industrial application of AOP was Microsoft Transaction Server and Enterprise Java Beans which were specific to a certain category of business applications – the middleware components. There was lack of any universal AOP framework which would cater to all requirements. Though there were couple of open source initiatives like AspectJ, Spring AOP etc, their adoption within enterprise was not widespread due to lack of support and organizational policies. (For example some enterprises would restrict use of open source tools due to complications arising out of support)
- ***Greater learning curve****:* Though AOP scored well on the separation of concerns part, adoption was still low owing to the complexity in the learning part. Weaving aspects within business functionality required experienced professionals who understood not only the nuts and bolts of AOP, but also how to use them optimally. Also trouble shooting an aspect oriented code required more insight into the framework which is being used. Scarcely documented frameworks were difficult to adopt by bigger programming community.
- ***Lack of adequate Literature:*** Though AOP has been received with enthusiasm to the researcher community, there is lack of adequate literature. The frameworks implementing AOP had their own documentations about the framework whereas standards and literature concerning AOP as a programming paradigm was not that developed. However its predecessors like OOP were more mature in terms available literature.
- ***Difficulty in quantifying benefits of AO implementation:*** Whenever outsourcing companies go through any new technology or paradigm, they need to justify the quantifiable benefits to the business. Although AOP as a concept exists for quite some time in the programming arena, there seems to be lack of standard tools / frameworks which can quantify the benefits of optimal adoption of AOP in any project.
- ***Legal Issues:*** Aspects could be a security risk because of them having control over all join points in the system. Program modules dealing with sensitive data like payroll, financial transactions when penetrated by aspect code pose security problem and violate regulatory compliance norms like SOX and BASEL II. In this context AOP needs to move from simple join points to complex encapsulation and security models to make sensitive join points unavailable for point cuts. [24]
- ***Multi Vendor Integration:*** In case of multi vendor projects, integration of aspect code becomes a challenge. In absence of a clear cut specification on the AOP tools, frameworks and methodologies, integrating AOP modules can be challenging. Moreover one aspect code cutting through other vendor's module can often result in confusion and yield undesirable results.

V. CONCLUSION

Aspect Oriented Programming is a topic of research for more than a decade over now. As per the MIT technical review in January / February 2001, AOP was one of the ten emerging areas of technology that will soon have a profound impact on the economy and on how we live and work [17]. However after almost a decade, the adoption of AOP in services industry has been very cautious and scarce. Also this is a research area which has received little attention from the researcher community. Adoption of AOP has been observed to a moderate degree in the IT products sector. However, penetration into the services sector is still scarce / unknown. There has been lot of work done in the area of quantifying the quality metrics AOP augments, but there is almost no research to find out the outsourcer's perspective in adopting AOP within the enterprise.

MS Transaction server and Enterprise Java Beans which were the first major industrial application of AOP still are not used to the extent it is expected where penetration of AOP would be optimal and meaningful. Complex enterprise applications are still developed without using Java EE features like container managed transactions, security and so on. Outsourcers seem to be concerned with the end product and not the means.

There are few success stories on AOP adoption by enterprises like Motorola [25], Siemens [26], HP [27] and SAP [24]. However large scale adoption of AOP in enterprise wide computing is either scarce or unknown. This needs to be investigated and corrective measured suggested so that AOP adoption will be optimum.

REFERENCES

[1]. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241, June 1997, 2-3

[2]. Cristina Videira Lopes, Aspect Oriented Programming: An Historical Perspective - Institute of Software Research, University of California, Irvine, December 2002, 9-10

[3]. Gregor Kiczales, John Lamping, Anurag Mendhekar RG: A case study for Aspect-Oriented Programming, February 1997,12-13

[4]. Karl Lieberherr and david H. Lorenz - Coupling Aspect-Oriented and Adaptive Programming April, 2003, 1-2

[5]. John Erwin, Jean-Marc Loingtier, John R. Gilbert, Gregor Kiczales, John lamping, Anurag Mendhekar, Tatiana Shpeisman - Aspect-Oriented Programming of Sparse Matrix Code –- December, 1997, 2-7

[6]. Gregor Kiczales, Eric Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold - An Overview of AspectJ — June, 2001,1-9

[7]. Xin Ma, Lian-he Yang -AOP Research based on Enterprise Application – World Academy of Science and Technology, 2008 [8]. Roger T. Alexander, James M. Bieman - Challenges of Aspect Oriented Technology – Workshop on Software Quality, Florida, 2002, 1-3 [9]. Jianjun Zhao -Change Impact Analysis for Aspect Oriented Software Evolution – Department of Computer Science and Technology, Fukuoka Institute of Technology – 2002, 3-5

[10]. Avadhesh Kumar, Rajesh Kumar, P.S. Grover - Maintainability of Aspect Oriented Software Systems – 2006, 6-7

[11]. Joon-Sang Leea, Doo-Hwan Baeb - An aspect-oriented framework for developing component-based software with the collaboration-based architectural style, August 2002, 7-9

[12]. Roberta Coelho,Arndt von Staa, Uirá Kulesza, Awais Rashid, Carlos Lucena Unveiling and taming liabilities of aspects in the presence of exceptions: A static analysis based approach – June 2010, 1-4

[13]. Jianjun Zhao - Maintenance Support for Aspect-Oriented Programs: Opportunities and Challenges – 2008, 2,6

[14]. Gail C. Murphy, Member, IEEE Computer Society, Robert J. Walker, Student Member, IEEE, and Elisa L.A. Baniassad - Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming, August 1999, 1-4

[15]. Cristina Videira Lopes, Martin Lippert - A Study on Exception Detection and Handling Using Aspect-Oriented Programming – 2000, 10-11 [16]. Freddy Munoz, Benoit Baudry, Romain Delamare, Yves Le Traon - Inquiring the Usage of Aspect-Oriented Programming: An Empirical Study – 2009, 2-4

[17]. http://www.ccs.neu.edu/research/demeter/aop/publicity/mit-tech-review.html - visited on 10-Dec-2012

[18]. Uirá Kulesza Cláudio Sant'Anna, Alessandro Garcia, Roberta Coelho, Arndt von Staa, Carlos Lucena - Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study – 2008

[19]. J. Kienzle, R. Guerraoui - AOP: Does it Make Sense? The Case of Concurrency and Failures.- Proc. ECOOP'02 – 2002, 2-5 [20]. The Open Group Architecture Framework (TOGAF) – Version – 9.0 [21]. Paulo Merson - Using Aspect-Oriented Programming to Enforce Architecture – Software Engineering Institute - 2007 [22]. http://www.ibm.com/developerworks/java/library/j-aopwork15 - Ramnivas Laddad, Feb 2006 - Visited on 01-Jan-2013

[23]. Kim Mens and Tom Tourwe - Evolution Issues in Aspect-Oriented Programming –

[24]. Christoph Pohl, Anis Charfi, Wasif Gilani, Steffen Göbel, Birgit Grammel, Henrik Lochmann, Andreas Rummler, Axel Spriestersbach - Adopting Aspect-Oriented Software Development in Business Application Engineering - AOSD Industry Track 2008 Brussels, Belgium [25]. T. Cottenier, A. van den Berg, and T. Elrad. The Motorola WEAVR: Model Weaving in a Large Industrial Context. - AOSD, 2007, 9-10

[26]. D. Wiese, R. Meunier, and U. Hohenstein. How to Convince Industry of AOP- AOSD, 2007,7-8 [27]. M. Mortensen and S. Ghosh. Using Aspects with Object-Oriented Frameworks - AOSD, 2006, 1-4