Aspect-Oriented Programming

Ekaterina Grekova, Gonzalo Pineda Zuniga, Grismika Gupta, Jamie Goldring, Mike Nguyen

***Abstract*—Aspect-Oriented Programming (AOP) has remained a controversial idea, for over decade. Its main objective is to improve the software development process and separate concerns using aspects. In this paper, AOP is compared to OOP. Both methodologies have their relative strengths and weaknesses that may influence the function of certain systems, in different ways. There are theoretical, implementation and complexity differences between AOP and OOP. In order to illustrate the effectiveness of AspectJ, the AspectJ tutorial is included in this report.**

I. Introduction

There are various software development methodologies that assist developers in structuring, planning and controlling their development. AOP is one of these, albeit controversial, software methodologies, introduced in 1996; it has since remained a controversial idea. There are not many studies that have been conducted on the benefits of AOP. As the case with any methodology, it is aimed to improve software development. AOP is a better fit than OOP with real domain problems for certain applications. Also, there are existing systems that have AOP-like properties.[3] Software development involves different tasks and activities; design and implementation processes, where various domain-related decisions made, are one of the major tasks. AOP allows easy capture of those decisions, thus resulting in increased readability and reduced debugging time. Aspects with certain decisions are clearly expressed and isolated, which leads to ease of code reuse.

Using AOP methodology, the design of an application consists of components, aspect languages, as well as the relationship between components and aspects. Different concerns are placed in components and aspects. Different abstraction and composition mechanisms can also be applied.[3] The design is aimed at separating code units, which simplifies development and maintenance of the application under development. Aspects address cross-cutting concerns that, in other methodologies, are time consuming and tedious. Any concern can be expressed by an aspect that is integrated into the system. With this approach, the concern is documented and isolated.[4] The developer's effort is reduced, and it reduces chances for defects in the application.

Design of an application largely depends on a chosen programming language. Different architectures can be appropriate for the same application but involves the use of different programming languages. AOP requires a component language, aspect language and aspect weaver for merging the languages. Each language has its characteristics and benefits. Since components document a procedure, procedural languages are appropriate; aspects document a concern or observation and require a procedural language as well that allows simple operations on nodes.[3] In addition, the role of software security is significantly increasing with time. It is a growing part of developers responsibilities. Originally security issues were not planned to be dealt with in AOP. Although, the base paradigm of AOP is promising for handling the hardening of software security practices. [2]

AOP is intended to improve software development by introducing aspects. Aspects document design decisions and are easy to reuse and change. However, it is difficult to quantify the benefits of AOP due to the lack of studies. As any new methodology, it should be thoroughly investigated and studied in order to prove that AOP is worth applying in small and large software development processes. Nevertheless, it is a promising methodology for the software development.

II. History of Aspect-Oriented Programming

A. Overview

The concept of Aspect Oriented Software Design (AOP) and the concurring development has several roots and direct antecedents which stem from known methodologies such as Subject Oriented Programming and Demeter/Adaptive Programming. [10] Aspect-Oriented Programming had actually been used for several years by the Demeter Team before the actual term was officially coined. It was not until 1997 that the explicit definition of Aspect-Oriented Programming was revealed. Gregor Kiczales, John Lampin and their colleagues at Xerox PARC officially introduced the definition at a Object Oriented Programming conference named ECOOP97. [3] This was the official and explicit birthplace of AOP. Starting with the early origins and definition of AOP lends to a better understanding of the historical perspective on how AOP has evolved since then, the landscape and current situation of AOP today and what ultimately lies ahead in the future.

B. Early Definition of AOP

The work done by the researchers at Northeastern University contain evidence of early instances of AOP. Northeastern also attempted an early general definition of AOP by abstracting from several known instances. Gregor Kiczaless definition of Aspect-Oriented programming has many similarities to the most general definition of AOP [11]. To summarize, the AOP book contains the definition of AOP but using a different terminology. Furthermore, it is important to stress that the Xerox PARC definition of AOP has changed over time. For a brief period, AOP was aimed only to deal with systemic aspects. However, the actual definition has now evolved to a point where it is more in line with the original definition found in the AP Book.

C. Mapping AOP to Hype Cycles, from birth till now

Hype cycles are generally used to characterize enthusiasm or "hype" and subsequent disappointment that typically happens with the introduction of a new technology. [16] There are 5 major phases to the hype cycle: technology trigger, peak of inflated expectations, trough of disillusionment, slope of enlightenment, and plateau of productivity[18]. Since its release, AOP has gone through almost all of these phases.

1) Technology Trigger: The first phase of a hype cycle which involves a breakthrough or product launch. The 2002 release of AspectJ 1.0 followed by a more stable and focused 2003 release of AspectJ 1.1 was the technology trigger. At the time, people in the programming field were captivated by the potential for AspectJ, and especially for enterprise applications. [18]

2) Peak of Inflated Expectations: The phase that typically involves a frenzy of publicity which in turn generates over-enthusiasm and unrealistic expectations. The peak of AOP occurred around 2004. Smaller companies outnumbered larger companies when it came to actually using AOP for their projects. The majority of developers working with AspectJ were from IBM. The peak of AOP itself did not stem from the inability to meet expectations but from the fact that AOP was always known as the 15

3) Trough of Disillusionment: The period of time following a peak where interest wanes as experiments and implementations become unfashionable and in some extremes fail. AOP had encountered this phase around 2006. A few factors lead to this inevitable trough. The main factor was due to the fact that a higher threshold of dedication was needed. Furthermore, AspectJ was perceived to have a steep-learning curve, due to its fresh syntax and Eclipse. AspectJs main IDE was also advancing at a rate that AspectJ itself had trouble following. The second determining factor was the serious competition that AspectJ had to face. Framework-centric approaches such as Enterprise Java Beans (EJB) and dynamic languages such as Ruby offered alternative solutions to tackle cross-cutting concerns. [18]

4) Slope of Enlightenment: The case where more instances of how the technology can benefit the enterprise start to crystallize and become more widely understood. [16] Developers are now showing greater signs of curiosity in hopes to seek further benefits and unleash new discoveries. The so-called complexity of AspectJ is now being embraced. This is due to several factors: the AOP team exemplified a positive response to user needs, followed by stable growth mainly due the fact that they had followed a path which allowed for their technology to be gradually introduced and revised. The acceptance of annotations and the disillusionment from the alternative technologies further kept AOP healthy. [18]

5) Plateau of Productivity: When the technology

becomes stable and is widely demonstrated and accepted. Mainly being used in situations where it is known to work successfully and productively, Java has already reached this plateau. It can be predicted that AOP and AspectJ may reach their peak in the upcoming few years. [18]

III. Overview of AspectJ

A. Features of AspectJ

AspectJ is a byte-code weaving AOP language which allows separating concerns that comprise a system. AspectJ comes from research work at Xerox Palo Alto Research Center, and has Gregor Kiczales as the project leader. He is the author of many of the sources used in this paper, and a recurrent name in in the studies of AOP. The goal of AspectJ is to make AOP available to a large number of developers, and they have aimed to reach that goal through the seamless integration of the language to Java. The IBMs Eclipse project provides ajc, a compiler that compiles both the aspect code and primary code. The ajc compiler adds to the ability to compile the AspectJ-specific keyword into byte code and facilitates the weaving of the byte code into class files. The aspect code is converted from an aspect construct into a class, and the other AspectJ-specific construct are converted to standard Java. [20] Some crosscutting elements of AspectJ are: Joint point: identifiable point in the execution of a program, it can take any form, such as the call to a method or setting the value of a variable Point-cut: program construct that captures the context generated at a certain joint point Advice: code to be executed at a certain joint point, we can choose if to execute before or after the joint point Introduction: static crosscutting instruction that can change classes, interfaces and aspects of the system Compile-time declaration: crosscutting instruction that throws warnings and errors at compile time if certain usage of patterns has been detected Aspect: main element of AspectJ, they contain the all the code for crosscutting concerns, and bring together the elements listed above into one unified Aspect, in a similar behavior as class does with its subclasses, methods and variables.[21]

B. Effectiveness of AspectJ

A number of articles have been encountered that stated the advantages of AspectJ. It can be concluded that effectiveness of AspectJ highly depends on domain it is applied for. Some applications can greatly benefit from using AOP rather than from any other programming practice. In order to determine effectiveness of AspectJ, it is important to compare applications that are developed for the same domain and to solve similar problems. For some applications, AspectJ can be one of the best options. Its effectiveness can be determined by all advantages of AOP, as described in this paper. However, AspectJ has some limitations. There are articles that compare Aspect-Oriented languages (like Java) to the Go-To statement, and point out the problems that are associated with Go-To statements and AOP. One of the similarities is that AspectJ introduces obliviousness of application. [12] When looking at the source code for an aspect, there could be an insufficient amount of information to determine a value of a variable. That variable could have been changed in a way that is not obvious for the developer. A system with aspects can be evaluated in terms of the allowed quantifications types, the purpose of the asserted actions, and the techniques for combining base-level actions with asserted actions. [13] AspectJ can be very beneficial for some systems, but not the others, as is the case with other programming techniques. It is important to analyze the domain and its problem in order to determine whether AOP and AspectJ, in particular, would be effective or not.

C. Security of AspectJ

From a security point of view, AspectJ is a programming language that has not been engineered with security as its first priority. [15] Despite this shortcoming, AspectJs current constructs are actually fairly useful for security hardening and is also considered one of the best languages that enforces security issues. The fact that aspect-oriented languages have been created to explicitly deal with the separations of concerns justifies why it is the appropriate choice for this situation. From a security engineer or programmer's perspective, the most convenient approach for them would be to inject and strengthen security without trespassing into logic territory of their application or software. Aspect-oriented progamming allows them to do this very well. Several security-relevant cross-cutting concerns are found scattered throughout the application logic: Logging Access control Error handling Transaction management Session management (in some cases) In aspect-oriented programming, the main program would not encode secure information; rather, it would be moved into a separate, independent section of code and centralize them. We do not ignore the fact that there will exist cases where centralize is very difficult or not possible. [14] Object-oriented programming (OOP), has the ability to separate concerns by grouping them into objects, however this comes with a disadvantage as it does not map well to abstract notions. Here is a general example of how object-oriented programming handles security compared to aspect-oriented programming: In OOP, we could write a central class for some application that checks all important sections of code when called. However, if one important check is forgotten in a vital area of code, the whole class would fail and would have no option for recovery. Why does this happen? Quite simply because of how this system was designed in an object-oriented fashion; security is a concern that will affect the entire system. Conversely, AOP can solve this problem with applying security concerns modularly through the entire system. By using aspect-oriented techniques, security policies can be separated and modified without affecting the actual code itself. This allows for developers to write the main application layer and potentially allow a security expert to handle security-related characteristics. However, this technique does not abstract the fact that a solid understanding of the aspect is still required in order to apply it properly. This was just a very general overview of how AOP handles security and much more discussion is required to actually illustrate how AOP enforces security issues successfully. One definite thing can be concluded however, and it is that programming in application-security looks promising and the potential uses are enormous.

Please see Appendix A, for a tutorial that has been designed to allow for easy understanding of how to use AspectJ. [7][8][9]

IV. Differences between AOP and Obect-Oriented Programming

A. Brief Intro to Object Oriented Programming

Object Oriented Programming (OOP) is a programming paradigm that came about, as a result of the limitations of procedural programming. OOP uses objects as a kind of data structure to represent real world problems. It allows the programmer to envision a problem in such a way that his/her solution will model and emulate a mini-world. This was a major shift from the paradigm at the time (procedural) which incorporated, in essence, top-to-bottom execution. In OOP objects are instantiations of classes, whereby a class is essentially a blueprint for an object. A program can instantiate 0 or more of these objects, as required, during execution. The objects can communicate with each other via methods, which receive data and perform specific operations. While these are the physical attributes that OOP brings to the table, the true power of OOP is the wide array of techniques that it utilizes. OOP provides programmers with techniques such as: encapsulation, modularity, polymorphism, and inheritance. Every programmer who uses OOP has encountered at least 2 of these techniques, and many have used them all.

B. Aspect-Oriented Programming vs Object-Oriented programming

1) Theoretical Differences: As with all paradigms, both AOP and OOP have their relative strengths and weaknesses. Typically the attributes of each paradigm are considered upon each situation that a programmer encounters. Various situations can arise when deciding what paradigm to implement for a software program: one can out-perform the other; one can perform as well as but in a simpler manner; one may out-perform the other in speed but at the cost of being far more complex to implement. Both OOP and AOP provide methods to encapsulate and group similar data to eliminate redundant and tangled or ”spaghetti” code. This provides a way of centralizing functionality, of which will ultimately provide a much more flexible architecture that is able to easy accept change.

For example: consider a large program that needs to be able to print a lot of data to the screen, and almost every class within the program will do this. Suppose at the outset of the project everyone agreed on a certain font, size, and colour. We know that this functionality is global to the program and can be centralized into one printing function. If the programmers on the project ignore this and put the printing functionality in each class, not only are they creating redundant code, but they are also going to have to change every occurrence of the print-code if the size, font, or colour changes. OOP provides the ability to create one printing function that can then be called from any class within the program. Any sort of change can be done in one place, hence the flexibility it provides. To view the differences between AOP and OOP, one must remember that AOP is an alteration and an extension of OOP [3]. As stated above, OOP provides the ability to group similar functionality together. It does so through classes which decompose the functionality. This is an excellent feature for information that can be decomposed in such a manner, but it is plain to see that not all functionality can be centralized in such a way. This is the key issue that AOP tries to solve.

AOP provides a higher-level view of implementation than OOP does. AOP separates functionality based on concerns rather than classes. This, in turn, allows for greater cohesion of functionality than OOP does. Consider the functionality of logging, where a program traces execution and keeps a record, which is primarily used for debugging. It has been proven that logging is a cross-cutting concern [5], which is a piece of functionality used in a system which cannot be cleanly decomposed in OOP. Since the functionality is required and OOP cannot decompose it, this leads to tangled or scattered code. AOP was designed purely to counter this problem. For one that has a firm background in OOP, the transition to AOP can be quite simple. To start, instead of considering classes as the way to separate functionality within program code, think ”aspects”; look down on the required functionality and assign each an aspect. Another cross-cutting concern is security. It is impossible in OOP to have a centralized security system that is flexible. Typically, security is scattered within program code, only appearing where it is required to be. In AOP we would see security as an aspect and thus contain all the security implementation there.

All of the above is the theoretical difference between AOP and OOP. It shows how AOP is based on OOP much the same as OOP was based on procedural programming.

2) Implementation Differences: Another difference, which may possibly be the most important, is implementation. Much the same as the implementation of OOP changed from procedural; AOP brings a whole new level of implementation to the table. The keywords and visual layout may vary from language to language, but the structure is very much the same. Gregor Kiczales, often called the father of AOP envisioned that the design of AOP would be even more modularized than OOP [3]. To a degree this is true. Teams of developers on one project can each be given an aspect to design, that in the end will be woven together, because all functionality is in one area; whenever a team needs functionality being developed by a different team, they simply weave that aspect in. This allows for minimal communication amongst teams, and a highly cohesive functional product at the end. This belief is core to AOP implementation, and by viewing the AspectJ tutorial (see Appendix A), you can see how it is done.

3) Complexity Differences: As with many computer science theories, the product may be less functional than the idea. The problem is the complexity introduced with AOP, during the implementation stage. In a regular OOP project, you would still have different teams working on different things, however the communication amongst teams must be effective and often to make sure that all of the teams products mesh. Some companies may even have teams of designers who design APIs/interfaces to make sure that the products function together as necessary. In AOP theory we shouldnt need any of this. We should be able to design our aspects and as long as we know the name of an aspect that we need to use in our own code, we shouldn't have to know anything about its implementation. In practice this doesnt happen [1]; we must understand the AOP code produced by the other team in order to coordinate our own functionality. The result is more cohesive, and the functionality such as logging is still maintained as a single aspect, but we still, as developers, need to know the behaviour of someone elses aspect. This is very difficult because of the complexity introduced by the AOP implementation: In AOP source code, you have a primary concern that you would convert into an aspect, which represents the basic functionality of the concern. During this process you have what is called ”unwoven” source code, because all of the aspects are independent, thus the concerns are independent. If OOP was the paradigm being used for this program at this stage, the resulting effect would be that there exists a number of independent classes that do not know about each other.

Fortunately, for OOP there are unit tests that can be utilized, which take those classes and monitor the way they function together. Unluckily for AOP, it introduces another stage before test can ensue. This stage is called the ”weaving” stage, where all the aspects need to be merged. At this point, the developers can see relationships between concerns and test how they affect each other. The reason testing cannot be done earlier is because of the fact that weaving concerns together induces dependencies that differ from testing the concern itself [1]. This, in turn, causes a degree of blindness on the part of developers, as the testing is pushed to a later phase. Keeping in mind that the emergent OOP ideas have been around since the mid 1960s, it can plainly be seen that it has had over 50 years to perfect its testing techniques! OOP initially did face similar criticism when people were shifting from procedural-style to OOP. In the beginning, OOP provided more theoretical properties at the cost of increasing implementation complexity, thus it would make sense that AOP would do the same.

V. Methods of Testing AOP

While the majority of this paper has focused on the functionality and theory that AOP brings to the table, this section focuses on the emerging testing concepts. One thing to keep in mind is that there is still a lot of information that is unknown with regards to testing specific parts of AOP [1]. Testing in general is becoming more prevalent in software engineering. Code bases are getting larger and more complex, and the need to get it right the first time, on top of maintenance, can make or break a company. Most of the standard testing principals that engineers follow as of today are all concepts tailored for OO programming. This is a result of the fact that OOP has been around for almost 50 years, and thus engineers have had 50 years to perfect their techniques. AOP is still a new radical idea that is gaining momentum, and so the need to refine its testing techniques is also gaining momentum. Since AOP is an extension of OOP, most of the techniques of testing being explored are also an extension of OOP techniques. To an extent, this method of extending the way we test OOP, to test AOP, has been accepted. The main issue is the lack of test support for the one major thing that AOP encompasses: Aspects. We are now at the point where we need a new testing feature that isnt used for OOP, to fully cover the tests needed to validate an AOP system. To show how an AOP system is tested, this section will discuss the correlating OOP test concept up to the point of testing woven code, while also illustrating the issues that make testing AOP so complex.

A. *The Unit Test*

In OOP almost every computer scientist has heard of and used Unit testing. It is fundamental to catch bugs as early as possible, and Unit testing is designed to do just that. To perform a unit test, one designs a Unit which is typically a Class, and tests its functionality by itself. Thus we test how the unit operates as opposed to testing how it interacts with other units. In AOP we do the same thing, except we must test concerns and aspects as units (remember that an Aspect is a concern, but a concern is not necessarily an Aspect). A concern that is not an Aspect is just ordinary OOP code that is tested using a standard Unit test. An Aspect is tested in a similar fashion; it is treated as a Unit, and its functionality is tested by itself. Thus, out of all the tests used in both OOP and AOP, the Unit test is almost identical. The key difference is the outcome of what we are looking for. In OOP we simply observe the functionality and make sure it functions. In AOP we look for two things: the advice and the pointcut. The advice is the functionality of the Aspect while the pointcut is the specification of the Aspect. If both the advice and the pointcut pass the test we move onto Integration testing.

B. The Integration Test

Integration testing is typically the phase of testing following the unit tests. In OOP it is the first test that checks the behaviour of the integrated Units. The Unit test covers each Unit individually, while the integration tests successively add Units and test their moulded functionality. This is a very important test phase for OOP, because while a Class may pass a Unit test, it may not cooperate with another class, resulting in bugs. In AOP integration testing is somewhat peculiar, because it tests how the Aspects and concerns function together, but it does this before the code is woven [19]. This is where the testing concepts diverge between OOP and AOP, and this gap that is created is the need for a new testing concept specifically for AOP. The gap is caused by the results of Integration testing for AOP as opposed to the results gained by Integration testing OOP. In OOP the goal of Integration testing is to determine the behaviour of units with other units. In OOP this is the result we get: The units either function together as we intend or they dont, and the developers make corrections until they do. In AOP, because the Aspects and concerns are tested prior to weaving, the way they behave together is not a clear indication of the system's behaviour. You may be asking yourself, if this is the case, why do we test prior to weaving? The answer: we have to test the behaviour prior to weaving so that we can compare the behaviour to post-weaving in an attempt to isolate bugs [1]. To skip the pre-weaving test we would be skipping a step in the test process.

C. Regression Testing

In OOP regression testing is performed on the system as a whole. It is done to ensure that as we keep adding units to our system, new bugs are not introduced. It is a tandem test to integration testing. We integrate the unit, run the integration test, and finally run the regression test. If this passes we repeat until weve done this for every unit. In OOP at this point we now have a system rather than individual units. This is radically different than regression testing for AOP. Regression testing in AOP is the testing on a system that has just been woven. In a way it is simply the integration test of the woven system. Thus the result is the behaviour of the woven system. It is called regression testing, because it is used to evaluate whether the weaving has introduced bugs. The problem is that the weaving is a complex process that can affect the system in ways unseen to the developers [1]. New tests will have to be written for each independent weave, because each weave can cause different results based on how it is woven. This is a huge problem because regression testing is supposed to see if added functionality introduces a problem. Weaving is not added functionality; it is just the method of integrating the aspects with the rest of the code. As such we dont know how the weaving is changing the behaviour like we do when we add functionality. This is a huge problem because how can you adequately test something that causes unknown changes? The fact is, regression testing does not work for OOP like it does for AOP.

D. The Missing Phase

In a sense, we are missing a test phase after integration testing. We need a test sequence that can allow us to see and test the weaving process to see how it affects the dependencies [19]. Mariano Ceccato suggested that we need to test AOP code using AOP tests; tests that are designed to be woven into the source that can trace the weaving process. The fact is that, right now, this is done through guessing and testing the weaving stage. This means that new tests are written to make sure the final system behaves as we want, and that the weaving is done over and over until the dependencies align as the developers want to. This is a huge waste of time and resources, when there is the potential for tests to be written into the weaving process. Unfortunately all of this is speculation driven by the need for more adequate testing. On the bright side, considering OOP has had 50 years to refine the testing, AOP has only been in development for roughly 15 years, meaning that we have 35 years to figure out this missing stage!

VI. Applications in Industry

Many tend to question the legitimacy of AOP since it is not commonly referenced to real-world industries. AOP has been designed to be implemented in many ways and has quite a few important uses. The key to its ongoing success has been the engineering and language, designed to make the language be more useable and the programs more easily deployable overall. Despite its limited exposure to the masses compared to other programming methods, AOP is being used quite extensively in many applications whether we know it or not. Web and application Servers, Enterprises, frameworks, monitoring tools and integration all implement AOP in one form or another. We will briefly take a look at how AOP is incorporated in each of these methods starting off with Enterprise applications. Enterprises are required to address many functions regarding crosscutting such as, transaction management, auditing, monitoring, concurrency control and error handling just to name a few. [18] Since AOP addresses these critical concerns, many enterprise applications use AOP. Furthermore, many programmers and developers discover that AOP can be used as an enabling technology for other projects they use, most notably Spring. It can be said that virtually every project which uses Spring, uses AOP since the Spring framework has a built-in AOP infrastructure. Industries in which AspectJ is used are: in or to aid production range from financial companies, health care and to various websites and web applications (e-commerce, content providers, etc.). The usage of AOP in Application frameworks can also aim to solve crosscutting concerns and improve functionalities while keeping overall structure well modularized. To further elaborate: Springs crosscutting solutions stem from transaction management and security through the use of aspects. In addition, a few frameworks have also slowly built-up using AspectJ as their foundation. A major strength of AOP or more specifically AspectJ, is that it makes a flexible monitoring scheme as easy and intuitive as it gets. Many monitoring tools make use of this advantage and use AspectJ as the fundamental technology behind their systems. Open source products like Glassbox, Perf4J and Contract4J all use AspectJ for different purposes. For example these tools are used to pinpoint bottlenecks, monitor applications or monitor contract violations. Commercial products also follow suit. SpringSource Application management Suite directs their focus to monitor application-level behaviours in Spring based apps. JXInsight and MantainJ are other commercial monitoring tools that provide awareness surrounding performance bottlenecks and system execution related tasks respectively. Lastly, the Eclipse project supports both command-line and Ant interfaces. The Eclipse IDE support (AJDT) for AspectJ and other providers of crosscutting structure is constantly being improved IDE support has been key to programmers using AspectJ and understanding crosscutting concerns. Although not as popular with the mainstream, AspectJ has been a standard tool for developers for many years and many years to come. Many consider it a secret-weapon in ways as it addresses many problems and offers extensive solutions to those problems, especially in enterprises as discussed earlier in this section.

VII. Future of AOP

According to Gregor Kiczales, the future of AOP will focus on the ironing-out of known problems related to low-level tools, as well as teaching techniques that will allow for even novice programmers to understand the way AOP works. Currently, advanced programmers are able to understand the techniques, however when tasked with teaching it to others, their methods are ineffective.[6] In the future, any advancements that are made with regards to AOP, the aim is to make these changes known and as easy to implement as OOP is right now.

Some parts of AOP implementation with AspectJ also need to be changed. Since it is a slightly newer language, compilers are not generally coded to understand it. [6] Advancements need to be made in this area soon, as well.

One type of AOP implementation that may be used in the future is the concept of Fluid AOP. With this, a programmer will be given an editable view of their program, and will be allowed to gather all places where a particular event or aspect occurs. Once put together, a programmer will be able to edit them all at once this will both make the process of debugging easier, as well as make the process of coding the program itself go a lot more efficiently. In the words of Gregor Kiczales, the ability would be there to twist the program into a new shape and edit it. [6] Though the program will still be a simple coded program, the notion that something like this would be possible is quite powerful, and the aim is to make it possible using Fluid AOP, in the near future.

VIII. Studies on AOP

Numerous studies and analyses have been conducted on AOP. Those studies aimed at identifying its applications, advantages and disadvantages. AOP is an independent programming style that can be applied on its own or along with other programming styles such as OOP. However, it still remains a controversial programming practice due to limited examples of industry AOP applications. Certain studies focus on identify what makes a language AOP. There are two properties that are necessary for AOP. Those properties are quantification and obliviousness. [13] Based on these properties, one can conclude if an AOP-like system is actually AOP-based or not, and why some domains are more suitable for AOP-based systems than others. Other studies mainly focus on identifying advantages of applying AOP to solve certain problems. AOP brings several advantages that include better approaches to solve certain domain problems as well as improvements of source code in terms of modularity and clarity. Despite the advantages described in different publications, disadvantages of AOP were revealed as well, as with any programming style. That is why it is important to analyse the domain and the problem that needs to be solved in order to select a proper programming style. A group of researchers took Dijkstra statement ...that the quality of programmers is indirectly proportional to the amount of Go To statements they use in their programs, to rephrase it and apply it to AOP: ... indirectly proportional to the amount of advice they use in their programs. [13] The future of AOP is uncertain, though goals are being aimed for; AOP has remained a controversial programming style for over a decade. However, the applications of AOP are limited. Aspects can be created only using few programming languages. That is not the only limitation of AOP. There are various articles available about limitations. One of them is The Real Costs of Aspect-Oriented Programming? by Roger Alexander. Researchers continue investigating AOP in terms of applications, advantages and disadvantages. However, there are only few industry examples of AOP-based applications. That is the main reason why AOP is still being studied and not used broadly.

IX. Conclusion

Aspect-Oriented Programming is a method of programming that, even after many years of study and use, is still being tested and improved upon today. It started off as an effort to increase the cohesion of and reduce the bugs that may happen within software programs that have many authors. The so-called creator of this programming method, Gregor Kiczales, hoped that it would theoretically increase the ability for a software programmer to isolate concerns that may arise within their system, and tackle them at a later time. It has gained popularity since its inception, because of its use of the concept ”modularity”. The term ”modularity” may be instinctively thought of as related to Object-Oriented Programming, however both methods of software organisation have very distinctive differences. AOP is, in effect, a variation or ”extension” of OOP itself, that aims to solve concerns within a system, that cannot be resolved with OOP. In theory, this would be a favorable tool that should be utilized regularly in software design, however in reality, program code that is organised using this method can become extremely complex. It still remains to be seen if this truly is a paradigm that could potentially be indispensible in the workplace.

X. References

[1] Alexander, R. (2003, November/December). *The Real Costs of Aspect- Oriented Programming?*: IEEE Software.

[2] Azzam Mourad, A. S.-A. (2009, January). *New Aspect-Oriented Constructs for Security Hardening Concerns.* Montreal, QC, Canada: Computers & Security.

[3] Gregor Kiczales, J. L.-M. (1997). *Aspect- Oriented Programming.* Palo Alto, CA, USA: Springer-Verlag Berlin Heidelberg.

[4] Pierre F. Baldi, C. V. (2008, October). *A Theory of Aspects as Latent Topics.* Nashville, Tennessee, USA.

[5] Edward Garson, Dunstan Thomas Consulting. *Aspect-Oriented Programming in C#/.NET*

[6] Darryl K. Taft, Gregor Kiczales(2007). *eWeek: The Father of AOP; Q&A:Gregor Kiczales talks about the evolution and the future of aspect- oriented programming*: ZIFF DAVIS MEDIA Inc

[7] Colyer A., Clement A., (2005). *Aspect-oriented programming with AspectJ*: IBM Systems Journal

[8] the AspectJ Team. ”Introduction to AspectJ Chapter 1. Getting Started with AspectJ.” Internet: http://eclipse.org/aspject/doc/released/progguide/starting-aspectj.html, 2003 [Feb. 24, 2012].

[9] the AspectJ Team. ”Chapter 1. Getting Started with AspectJ.” Internet: http://eclipse.org/aspectj/doc/ released/progguide/starting.html, 2003 [Feb. 24, 2012].

[10] Karl J. Lieberherr. ”Connections between Demeter/ Adaptive Programming and Aspect- Oriented Programming (AOP).” Internet: http://www.ccs.neu.edu/home/lieber/connection-to-aop.html, [Feb. 28, 2012]

[11] Karl J. Lieberherr, (1996). *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterms* PWS Publishing Company, Boston

[12] Constantinos Constantinides, T. S.-M. (2004, August). *AOP Considered Harmful.* Germany: Karlsruher Institute of Technology

[13] Robert E. Filman, D. P. F. (2001). *Aspect-Oriented Programming is Quantification and Obliviousness*: RIACS Technical Report 01.12

[14] John Viega, J. T. -C. (2001, February). *Applying Aspect-Oriented Programming to Security*: Cutter IT Journal

[15] Dima Alhadidi, N. B.-D., . *AspectJ Assessment from a Security Perspective* Montreal, Quebec: Concordia Institute for Information Systems Engineering

[16] Jackie Fenn. (2008, June). ”Understanding hype cycles: When to Leap on the Hype Cycle - Gartner Group”. Internet: http://www.gartner.com/pages/story.php.id.8795.s.8.jsp, [April 1, 2012]

[17] John Reitano. ”The 15% Solution.” Internet: http://www.drdobbs.com/architecture-and-design/184414845, [April 1, 2012]

[18] Ramnivas Laddad. ”AspectJ in Action.” Internet: http://manning.com/laddad2/excerpt_perspectiveAOP.html, [April 5, 2012]

[19] Mariano Ceccato, P. T.-R. (2005). *Is AOP code easier or harder to test than OOP code?* Centro per la Ricerca Scientica e Tecnologica 38050 Povo (Trento), Italy: ITC-irst

[20] Joseph D. Gradecki, Nicholas Lesiecki (2003). *Mastering AspectJ: Aspect-Oriented Programming in Java* John Wiley & Sons, Inc., New York, NY, USA.

[21] Ramnivas Laddad (2003). *AspectJ in Action: Practical Aspect-Oriented Programming* Manning Publications Co., Greenwich, CT, USA.

APPENDIX A

AspectJ Tutorial

AspectJ is an implementation of aspect-oriented programming for Java. AspectJ adds to Java just one new concept, a join point, and it only adds a few new constructs: pointcuts, advice, inter-type declarations and aspects (source: the AspectJ Team. "Introduction to AspectJ Chapter 1. Getting Started with AspectJ." Internet: http://eclipse.org/aspject/doc/released/progguide/starting-aspectj.html, 2003 [Feb. 24, 2012]).

A crosscutting concern is a single concern in the design or implementation of a system that impacts multiple places in the static structure of the system or in its runtime control flow. (source: Colyer A., Clement A., (2005). Aspect-oriented programming with AspectJ: IBM Systems Journal)

Many software developers are attracted to the idea of aspect-oriented programming but they do not know in which way to approach it (source: the AspectJ Team. "Chapter 1. Getting Started with AspectJ." Internet: http://eclipse.org/aspectj/doc/ released/progguide/starting.html, 2003 [Feb. 24, 2012]). Here is our tutorial.

Requirements

- Eclipse SDK Version 3.7.1
- Eclipse AspectJ Development Tools
- JavaSE-1.6

Creating a Java Project with Eclipse

1. Go "File => New => Java Project"
2. For project name, just use "Tutorial"
3. Remember to choose "JavaSE-1.6" under "Use an execution environment JRE:" (it has been largely reported on the Internet that the latest version of java, JavaSE-1.7 and AspectJ present some issues with compatibility)
4. Go to the "Package Explorer", under "Tutorial" right click on "src", and select "New => Package", in "Name", just use "seng403"
5. Now we have a Java project "Tutorial", and a package to work with the project "seng403"

Creating some Java classes

Start by creating two normal Java classes: Point and Line.

- Go to the "Package Explorer", under "Tutorial => src", right click in "seng403", and select "New => Class", a window will pop-up, and here just name it "Point", and hit the "Finish button"
- Make sure your Point.java class contains the following:

Point.java class

```java
package seng403;

public class Point {
	private int _x, _y;

	public Point() {
		_x = 0;
		_y = 0;
	}

	public Point (int x, int y) {
		_x = x;
		_y = y;
	}

	void setX(int x) {
		_x = x;
	}

	void setY(int y) {
		_y = y;
	}

	int getX() {
		return _x;
	}

	int getY() {
		return _y;
	}
}
```

Now to create your Line class:

- Go to the "Package Explorer", under "Tutorial => src", right click in "seng403", and select "New => Class", a window will pop-up, and here just name it "Line", and hit the "Finish button"
- Make sure your Line.java class contains the following:

Line.java class

```java
package seng403;

public class Line {

	private Point _p1, _p2;

	public Line() {
		_p1 = new Point(0,0);
		_p2 = new Point(1,1);
	}

	public Line(Point p1, Point p2) {
		_p1 = p1;
		_p2 = p2;
	}

	void setP1(Point p1) {
		_p1 = p1;
	}

	void setP2(Point p2) {
		_p2 = p2;
	}

	Point getP1() {
		return _p1;
	}

	Point getP2() {
		return _p2;
	}

	public String toString() {
		return "(" + _p1.getX() + "," + _p1.getY() +
			") to (" + _p2.getX() + "," + _p2.getY() + ")";
	}
}
```

Now we can create a third Java class, "Tutorial", that will have the implementation of the main method:

- Go to the "Package Explorer" and under "Tutorial => src", right click in "seng403", and select "New => Class", a window will pop-up, and here just name it "Tutorial", and hit the "Finish button"
- Make sure your Tutorial.java class contains the following:

Tutorial.java class

```
package seng403;

public class Tutorial {

	public static void main(String[] args) {
		Point p1 = new Point(0,0);
		Point p2 = new Point(1,1);
		Line l1 = new Line(new Point (0,0), new Point (-1,-1));
		System.out.println("P1 x coordinate = " +
					p1.getX() + ", y coordinate = " +
					p1.getY());
		System.out.println("P2 x coordinate = " +
					p2.getX() + ", y coordinate = " +
					p2.getY());
		System.out.println("L1 coordinates = " + l1.toString());
		p1.setX(-2);
		p1.setY(-2);
		l1.setP2(p1);
		System.out.println("P1 x coordinate = " +
					p1.getX() + ", y coordinate = " +
					p1.getY());
		System.out.println("L1 coordinates = " + l1.toString());
	}

}
```

The above three classes, are nothing more than regular Java classes that manipulate Points and Lines.

Creating an AspectJ Project with Eclipse

What we are going to do now, is to transform our currently Java project, to an AspectJ project, this can all be done in Eclipse, in one step:

- Go to the "Package Explorer", and right click on the "Tutorial" project, and select the "Configure => Convert to AspectJ project" option

Our "Tutorial" project is now an AspectJ project, but it is missing aspects, so let's go onto creating one.

Creating an Aspect

In this tutorial, we are going to create an aspect that tracks movement of our objects, ie. every time a point or a line gets re-set, we print to console that a movement has taken place.

With this simple move tracking class we want to illustrate the power behind AspectJ. With a relatively simple aspect creation, we can put a monitor on our classes, without the need of any modification to our original java classes.

In order to create an aspect we have to:

- Go to the "Package Explorer" and under "Tutorial => src", right click in "seng403", and select "New => other", a window will pop-up, go under "AspectJ" folder, and select "AspectJ" icon, after hit "Next", and here just name it "MoveTracking", and hit the "Finish button"
- Make sure your MoveTracking.aj aspect contains the following:

MoveTracking.aj

```
package seng403;

public aspect MoveTracking {

	pointcut moves() :
		call (void Point.setX(int)) ||
		call (void Point.setY(int)) ||
		call (void Line.setP1(Point)) ||
		call (void Line.setP2(Point));

	after() : moves() {
		System.out.println("Something has been moved!");
	}
}
```

Here our pointcut moves(), will trace any call to our set methods in our package, and the after() : moves, will execute any code we would like after the calling of this methods.

Testing our AspectJ Project

To execute our project, we go to the "Package Explorer", then under "Tutorial => src => seng403", right click on "Tutorial.java", and select "Run As=> Java Application".

We should be able to see the following output:

```
P1 x coordinate = 0, y coordinate = 0
P2 x coordinate = 1, y coordinate = 1
L1 coordinates = (0,0) to (-1,-1)
Something has been moved!
Something has been moved!
Something has been moved!
P1 x coordinate = -2, y coordinate = -2
L1 coordinates = (0,0) to (-2,-2)
```

Small modification to our AspectJ Project

We could also, modify our example above, to a more proactive aspect, that detects when something will get moved, check this different version of MoveTracking.aj:

```
package seng403;

public aspect MoveTracking {

	pointcut moves() :
		call (void Point.setX(int)) ||
		call (void Point.setY(int)) ||
		call (void Line.setP1(Point)) ||
		call (void Line.setP2(Point));

	before() : moves() {
		System.out.println("Something will get moved!");
	}
}
```

We should be able to see the following output:

```
P1 x coordinate = 0, y coordinate = 0
P2 x coordinate = 1, y coordinate = 1
L1 coordinates = (0,0) to (-1,-1)
Something will get moved!
Something will get moved!
Something will get moved!
P1 x coordinate = -2, y coordinate = -2
L1 coordinates = (0,0) to (-2,-2)
```

AOP addresses the problem of cross-cutting concerns, which would be any kind of code that is repeated in different methods and can't normally be completely refactored into its own module, like with logging or verification. So, with AOP you can leave that stuff out of the main code and define it vertically like so: Function mainProgram() {. Var x = foo(); doSomethingWith(x); return x; }. Aspect logging {. Before (mainProgram is called): {. Log.Write("entering mainProgram"); }. After (mainProgram is called): {.