Collapsible Tabular Visualization of Aspects in Object Oriented Programming

Md Nahid Rahman, Md Naim Hossain, Young Lee
Department of Electrical Engineering and Computer Science
MSC 192, 700 University Blvd., Texas A&M University - Kingsville
Kingsville, TX 78363. USA
nahidrmn@gmail.com, findnaim@gmail.com, young.lee@tamuk.edu

Abstract - Due to its implicit invocation nature of Aspect Oriented Programming (AOP), locating a joint point for executing aspect is extremely difficult. Hence, it becomes difficult to understand the application's flow and behavior. Current AOP visualization tools have limitations such as high dependency on other tools, confusing and excessive use of color to represent aspects and using an outdated version of AspectJ. In this paper, we propose a new approach collapsible tabular visualization tool to visualize and represent AOP features to aid the programmers in better understanding AOP applications. We have come out of traditional color-based aspect visualization and developed a web based tool: AspectViz that visualizes the aspects in a simple collapsible table. A questionnaire containing four different questions related to aspects visualization was developed to compare AspectViz with current visualization tools. 20 graduate students and professional software developers were invited to participate in the test experiment as well as the survey. We have compared its performance with existing AOP visualization tools i.e. the AJDT and the AspectMaps and showed how it outperformed in many cases, which is no color confusion, simple tabular visualization of aspects, no dependency on third-party software, easy to understand and the time it took to find a particular aspect was less etc. Collapsible tabular visualization enhanced the usability and performance of aspect locating in aspect-oriented programming.

1.0 INTRODUCTION

Object Oriented Programming (OOP) can modularize a program at great extent, but there are some concerns in software systems that are not possible to modularize using OOP anymore. Aspect Oriented Programming (AOP) was introduced to provide absolute modularity in software development [1]. Some concerns such as logging, transaction demarcation, and management, application profiling and security are the features that spread all over the source code because of the nature of these features. AOP can modularize each of these concerns and implicitly plug inside the base code wherever needed. Since they are implemented implicitly, it is impossible to find out which aspect is operating at which part of the base code. AOP visualizer provides the facility to view the link between aspect and base code. Hence, it becomes easier to understand the flow and behavior of the program.

There is a separation between business concerns and crosscutting concerns while designing application. During the development phase, the concerns become tangled and break some of the basic software design principles such as Single Responsibility Principle (SRP) by implementing both business core and crosscutting concerns together inside a single class. We will discuss some of the disadvantages of not using AOP in the following:

Code tangling happens when multiple concerns are implemented in a single module. "Developers consider concerns for example business logic, application performance, flow synchronization, application logging, application security etc. when implementing a module" [2]. The following Figure 1 shows tangling in a class.

This drags to simultaneous appearance of elements originated from individual concern's implementation and finally turns into code tangling.

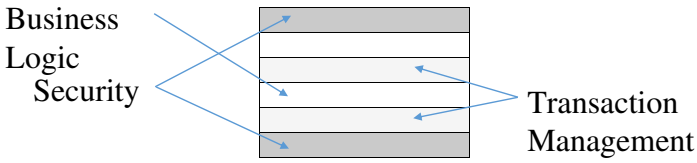

Figure 1: Code Tangling in a Class

The Code scattering happens if one functionality is written across several modules. By definition, crosscutting concerns usually spread over multiple modules, hence similar codes are also scattered within all these modules. For instance, if a software makes database calls, then the performance of a concern can affect the performance of other modules if those modules are also calling the database.

DOI: 10.5176/2251-3043_5.3.374

AspectJ is one of the most integral implementations of the AOP model and it supports all the elements. It offers two syntax choices: traditional keyword and @AspectJ annotation. We can categorize the crosscutting structures in AOP as a common crosscutting elements (such as join point, the pointcut, and the aspect), the dynamic crosscutting element (e.g., advice) and finally static crosscutting elements (inter-type and weave-time declarations). All of these elements also form the actual building blocks of AspectJ. A weaver is needed to weave the classes and the aspects together so that advice gets executed.

Even though AOP seems to express crosscutting concerns in an efficient and elegant way, it is still a relatively young programming paradigm and not much work has been done in defining and assessing the quality of aspect-oriented programs. Visualization of AOP project can help to improve the quality of software systems. Some groups of people have done research and study [3][4] on this issue. Here, we'll discuss various aspects of AOP visualization from different perspectives and propose a new approach to visualize the AOP project.

AOP visualization is an advanced and less explored field than other areas of software engineering and development. Different visualization group visualized AOP in different ways; in most cases, extensive color is being used. On the other hand, we have used collapsible tabular format and avoided any unwanted colors to free the developers from being confused. Some of the visualization used heavy and complex tool to analyze the source code. For example, AspectMaps uses moose reverse engineering tools but we relied on a basic and important resource only: the source code itself. We can make our own logic and can manipulate them accordingly.

2.0 LITERATURE REVIEW

There are some AOP visualizers available where each of them has adopted different approaches to performing the visualization task. Here, I will discuss two of the most prominent and widely used AOP visualization tools: AJDT and AspectMaps. Their tool showed a part of the AOP elements inside the source code, by considering the intention of the program developer. Visualization is considered as the UML extension which also represents aspects, advice and method call. The power of this approach is its ability to perform abstraction operations automatically, and automatic element selection which will be visualized.

2.1 The AJDT

The AspectJ Development Tools (AJDT) is an open source Integrated Development Environment (IDE) that supports programming with AspectJ within Eclipse. It highlights the syntax, reports error along with many other features that help the user about understanding the aspects in the program. AJDT provides aspects browsing capability same as class browsing capability. AJDT visualizer visualizes an entire project at a glance, integrated debugging support and wizards to create aspects and AspectJ projects. It can visualize any AOP projects and can be found from the perspective mode of the eclipse.

With AJDT, we can navigate through the aspects and advice of the AOP project but we cannot scale up the segments to have a more clear view. Each column is the representation of AOP source file and the colored portions of it are the advice. Each color represents different advice which is shown in Figure 2.

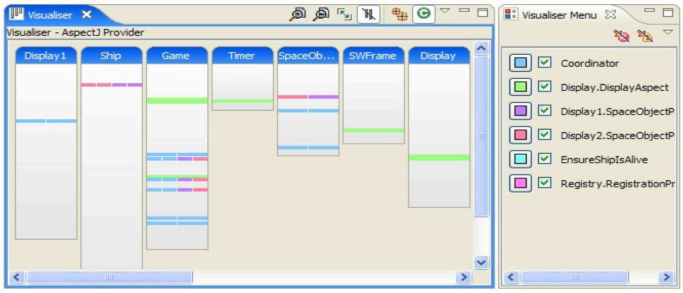

Figure 2: AJDT Representing Aspects with Different Colors

2.2 The AspectMaps

Another well-known aspect visualization tool is AspectMaps. This tool extends the Moose reverse engineering platform by implementing various support tools from Moose [5]. One important feature of AspectMaps is its zooming facility of a selected part of source code. Zooming from a coarser level to fine-grained level exposes more details about the code. The visualization is localized, for example in one window for some packages. The level of visualization can be package level, but in a different window, it can zoom to class level for other packages [6]. The AspectMaps window is given below Figure 3.

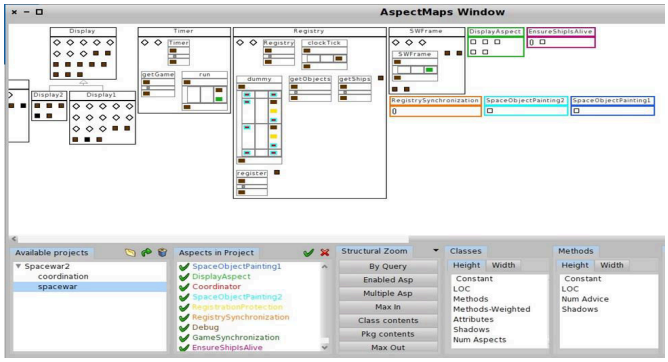

Figure 3: Structural Zooming of all Join Point Shadows of AspectMaps

When a project is opened with AspectMaps, the package content is put inside a rectangle with the name of the package on top of it. It colors all the rectangles by the color specified to the aspect of it. If there are multiple aspects, then the rectangle is colored with black. As we start to zoom-in to the package rectangle, it starts to show the details inside it such as classes and aspects, which aspect resides in which class with corresponding aspect's color. Below Figure 4 is the visualization of a sample AOP project showing package, class and method level AOP feature representation.

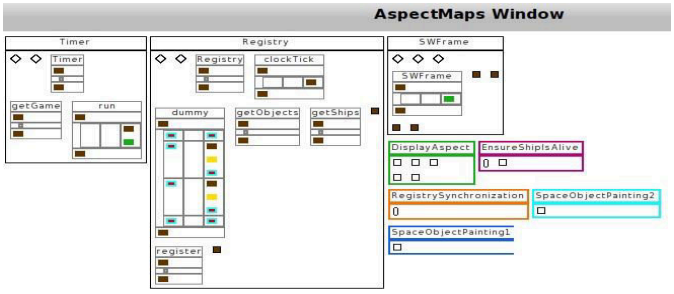

Figure 4: Visualization of AOP Project Using AspectMaps Visualizer

3.0 CHALLENGES AND PROBLEMS

AOP has higher quality as software systems, but it also causes some problem in understanding already developed software and its behavior. In the accompanying segment, we will talk about difficulties and issues that join AOP.

3.1 Challenges in AOP

AOP modularizes the cross-cutting concerns and plugs in these modules inside business logic wherever it is required. Inherently, it also increases the complexity of software to understand its architecture and behavior [7]. Since aspect codes are not physically placed where they are going to make the effect, it later becomes vague to the software developers to figure out which aspect is operating at which part of the base code. Hence, visualization of AOP features was advised by the experts. As a result, various approaches have been proposed and tools have been developed. Each tool and approach have its own shortcomings and hence there is a need for better and more effective ideas to comprehend the structure of an AOP application by visualizing it. We have to keep in mind that the ultimate goal of aspect visualization is to know where an aspect is advising a join point, no matter which approach we take to visualize it.

3.2 Problems in Existing Visualizers

To effectively show how an advice is executing at which join point, several groups have proposed and developed AOP visualization tools. Few of them have been successful and popular with software developers. Most of them have some problems and difficulties to properly visualize the software. For instance, AJDT and AspectMaps are most popular visualizer for AspectJ framework. One common problem both of these visualizers suffer from is they can visualize AOP programs that are written only in older AspectJ constructs i.e. traditional keyword based. But, after the release of Java 5, since it had support for annotations, people in the software industry were using annotations other than traditional AspectJ constructs. Hence, these visualizers cannot support newer AOP projects. The traditional AspectJ program requires a special compiler to compile the AOP source code, but annotation based AOP code can be compiled using the regular java compiler (javac).

3.2.1 Problems in AJDT

The most common difficulties faced when using AJDT aspect visualizer are becoming confused with the colors of aspects and keeping track of columns (source code files) in the visualizer main window. In AJDT visualizer, a unique color is assigned to each advice and as the number of advice increases, it becomes difficult for the programmer to remember which color belongs to which aspect. Another problem is that it shows each source class file as a column where aspects are advised. As the project becomes bigger, the numbers of classes where the aspects can be plugged become larger. Sometimes, it becomes several hundred classes which are very difficult to maintain in the visualizer window and to keep track of classes and aspects. This is an unavoidable maintainability issue for AJDT visualizer.

3.2.2 Problems in AspectMaps

The most obvious problem associated with AspectMaps is its usability. It doesn't directly work with the source code and to visualize a project it needs some extra processing and tools to mediate. Since it is an extension of the Moose reverse engineering tool, the source code needs to be made suitable for Moose tool. To do that, the source code is converted to .mse and .xcr file.

To import Eclipse projects into AspectMaps, two external tools are required: firstly, to generate Moose .mse files for the Object-Oriented part of the application and secondly to generate .xcr files for the Aspect-Oriented part of the application. These two tools are inFamix and VerveineJ and they are protected by license, not open source. The flow is shown on following Figure 5:

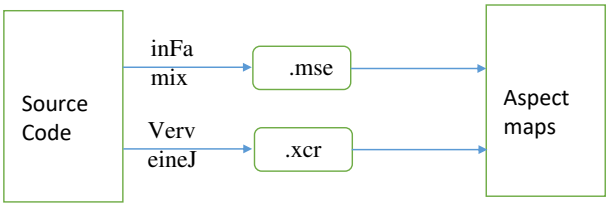

Figure 5: AspectMaps Workflow

This extra processing is a tedious job for a project because the source code is constantly changing for a project. Whenever the source code is changed, the .mse and .xcr files need to be generated every time. AspectMaps also suffers from the similar problem like AJDT: the confusion with color. In the visualizer window, AspectMaps doesn't show the name of the Join Point and Advice. To see those, we have to expand each class and then see the tool tips. This is a time-consuming job if the number of classes and aspect are high.

Another problem is that the border color of rectangles are colored with the same color of aspect with which it has a reference. If a class has more than one aspect reference, the class rectangle's border is colored black which doesn't make any significance. Hence, it's not possible to tell which aspect is executing inside that class.

4.0 PROPOSED APPROACH

A different approach is taken to sort out the issues and limitations. First of all, no color is used to specify the aspects. As the developers are familiar with a limited number of colors no confusion arises. Instead, tabular representation is used where aspect classes are listed in the left column and on the right side of table Pointcuts and Join Points are listed.

All the information related to aspects can found in a single place. Developers do not need to go back and forth between base code and aspect source code. Secondly, it is independent of development tools. All it needs is an aspect implemented with AspectJ framework. Third, the flow of visualization of these tools are from base code to aspect code i.e. first they find out the base code where aspects are operating, then they gradually navigate towards the aspect. On the other hand, our approach finds out the aspects available in the program first then finds out the operating base code (Join points). Even if the program isn't compatible, it works fine. It can be used as the documentation of source code as well; as it generates the Portable Document Format (PDF) that shows the changes in source code with the evolution of software.

4.1 Design and Implementation

To achieve our goal we have developed a web-based tool named AspectViz that uses JSP and Servlet. As it is web-based developers can access it from anywhere without installing any software or plugins. Moreover, AspectViz doesn't store any input source code that keeps the proprietorship of the developers without any violations. It takes any project i.e. developed with annotations based implementation of AspectJ. Feature-based extraction approach of the aspects is used which is described in the following Figure 6.

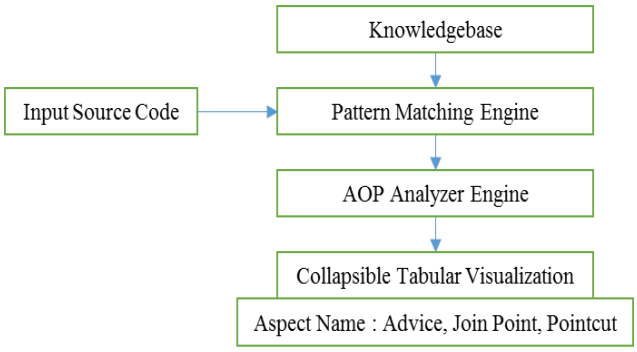

Figure 6: Design Components of AspectViz

4.1.1 AOP Knowledge Base

AOP features are limited; so, we created a knowledge base to store the AOP features of AspectJ. Knowledge base stores the AspectJ annotations and language keywords such as @Aspect, @Before, @After. Data structure hashmap is used to store them rather than database management systems. Knowledge base works as the input to the pattern matching engine and shown in Figure 7.

```
public class KnowledgeBase {
    public Map<String, String> getKnowledgeBase(){
        Map<String, String> knowledgeBase = new HashMap<String, String>();
        knowledgeBase.put("@Aspect", "aspect");
        knowledgeBase.put("@Before", "beforePointcut");
        knowledgeBase.put("@After", "afterPointcut");
        knowledgeBase.put("@Around", "aroundPointcut");
        knowledgeBase.put("@AfterReturning", "afterReturning");
        knowledgeBase.put("@AfterThrowing", "afterThrowing");
        return knowledgeBase;
    }
}
```

Figure 7: AOP Knowledge Base

4.1.2 Pattern Matching Engine

This engine is one of the cores and most important parts of AspectViz. It is used to locate AOP features in the project. Source code and knowledgebase are considered as input to this engine. It scans each source code file and searches for patterns in the knowledge base. We have implemented pattern matching using regular repressions and existing pattern matching java libraries. Then, we scanned the source code to match it with the features stored in the knowledge base. First, we searched for Aspects in the source files. Once found, we looked for Pointcuts specified in the Aspect class. Then, we found out the type of Pointcut, line number and the number of Pointcuts in an Aspect. After that, we extracted the name of the Advice method and the Join Point locations for each Pointcut. Hence, all the AOP features: Aspect, Pointcut, Advice and Join Point are discovered and shown in the output window. The flow chart of the pattern matching engine is shown in Figure 8.

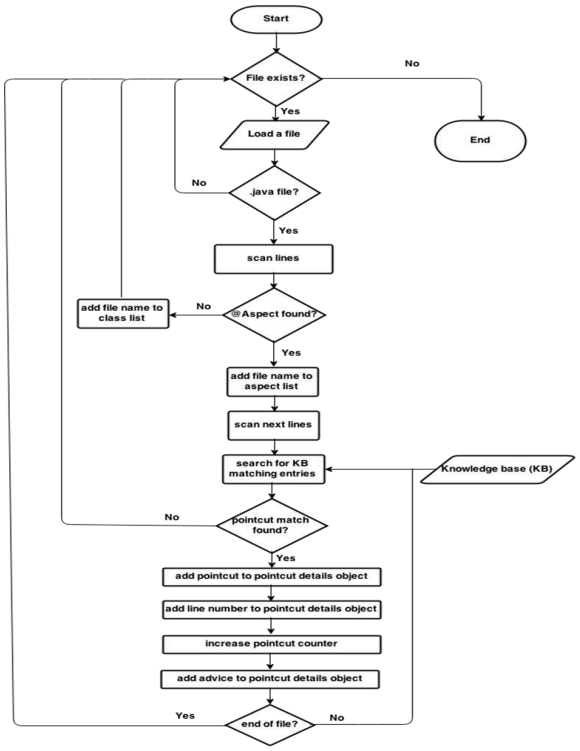

Figure 8: Flow Chart of Pattern Matching Engine

4.1.3 AOP Analyzer Engine

The pattern matching engine returns the matched component to AOP analyzer engine; then the analyzer engine decides the type of the component and finds out other related information for this component, such as for an aspect, to which package it belongs, the number of pointcuts it has, etc. It then organizes the features and displays them in a tabular format. We have listed the aspect and classes separately for the convenience of users. The following Figure 9 shows the flow chart of AOP analyzer engine:

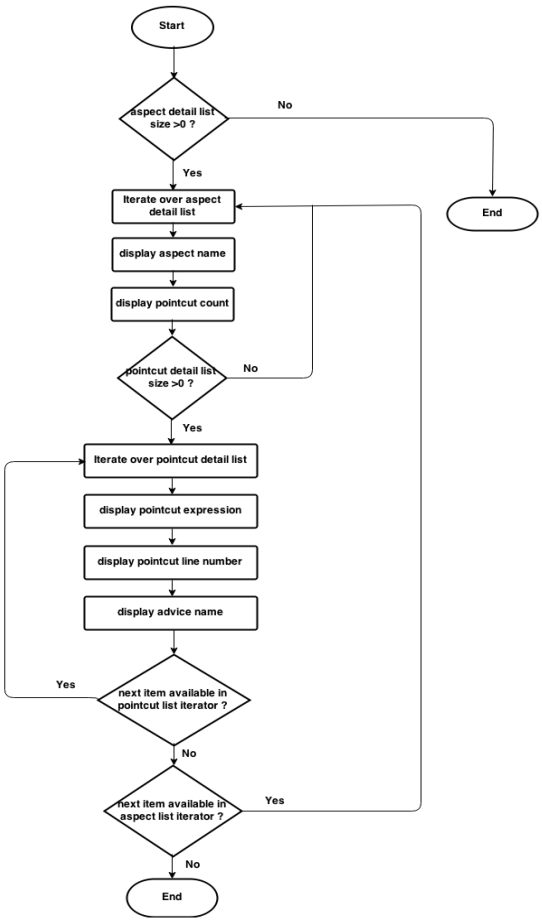

Figure 9: Flow Chart of AOP Analyzer Engine

4.2 Collapsible Tabular Visualization of Aspects

It is very hard to determine the aspects when the number of aspects is higher. When there are a lot of aspects to be visualized it necessary to scale down to a small segment of rows. Now, if we visualize the aspects by scaling down, it needs to be scaled up whenever necessary. So, we came up with a solution that can scale up to a full description and can scale to down to minimum information.

Our proposed idea shows a list of aspects rather than full description at the first. All the descriptions will be expanded whenever the description link is clicked. Collapsed or scaled down to a list of aspects is given in Figure 10. And the clickable expansion of the full description is illustrated in Figure 11.

Aspect	Pointcut	
Class: com.sortmeter.aspects.SelectionSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.BubbleSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.MergeSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.InsertionSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.QuickSortAspect.java	**Number of Pointcuts:** 2	Description

Figure 10: Scaling Down to a Small Portions Visualization of Aspects

Aspect	Pointcut	
Class: com.sortmeter.aspects.SelectionSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.BubbleSortAspect.java	**Number of Pointcuts:** 2 **1) Pointcut Name:** @Before("execution(* com.sortmeter.impl.BubbleSort.bubble_srt(..))") **Pointcut Type:** @Before **Line Number:** 22 **Advice:** public void logBeforeBubbleSort(JoinPoint joinPoint) **Join Point:** com.sortmeter.impl.BubbleSort.bubble_srt(..) **2) Pointcut Name:** @After("execution(* com.sortmeter.impl.BubbleSort.bubble_srt(..))") **Pointcut Type:** @After **Line Number:** 32 **Advice:** public void logAfterBubbleSort(JoinPoint joinPoint) **Join Point:** com.sortmeter.impl.BubbleSort.bubble_srt(..)	Description
Class: com.sortmeter.aspects.MergeSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.InsertionSortAspect.java	**Number of Pointcuts:** 2	Description
Class: com.sortmeter.aspects.QuickSortAspect.java	**Number of Pointcuts:** 2	Description

Figure 11: Scaling up and Showing Full Description of the Particular Aspect

5.0 EXPERIMENTAL METHOD AND RESULTS

To evaluate the performance and effectiveness of AspectViz, a survey is conducted. The sole purpose of this survey is to compare usefulness, simplicity, and performance of AspectViz and AspectMaps.

5.1 Experimental Method

A set of four questions is being compiled to answer the key features of AOP. Developers ask common questions related to aspects are what are the aspects, what do they do and where are they applied. Considering these facts, four questions are selected. Firstly, we demonstrated AspectViz, AspectMaps and test program to them. All the 20 participants were given a computer and questions printed on a paper. There were no time constraints to complete the survey. Each participant was asked to measure the time in seconds to answer each question and also rate the tools out of 10 based on their experience and comfortableness with the tools. To answer each question participants started from the homepage of the visualizer so that the results do not get affected by another process.

1. Find two aspects and their package names?
2. What are the types of each pointcut in MergeSortAspect aspect?
3. What is the advice inside MergeSortAspect?
4. How many join points is the aspect SelectionSortAspect operating on?

5.2 Experiment Results

From the result of the survey conducted, we have calculated the average time taken to answer each question and the average rating for each question on the tools. The following figure shows the time comparison and rating based on the survey result.

AspectMaps visualizes traditional non-annotation AOP of AspectJ. To test it we have taken sample application available on AJDT website named spacewar2. AspectViz can visualize annotation based AOP of AspectJ; Hence, we developed a project sort-meter-aspected that measures the execution time of various sorting algorithms for the same set of input data.

Here, the time to answer each question is measured in seconds and the rating is given out of 10. It's worth mentioning that, the values of these parameters are not absolute and may vary from applications to applications and the skill of the people in AOP doing the test. Survey result is given in the below table:

TABLE I: COMPARISON OF EXPERIMENT RESULTS BETWEEN ASPECTVIZ AND ASPECTMAPS

Question No.	AspectViz Time	AspectMaps Time	Rate AspectViz	Rate AspectMaps
1	22.8s	26s	9	8.8
2	11.4s	17.6s	9.5	8.4
3	12.8s	38.8s	8.8	7.2
4	12.8s	49s	9.6	7.4
Average:	14.95s	32.85s	9.2	7.95

Now, these 20 people are divided into two groups and each group consisted of 10 members. Another survey was conducted between two groups of people named CS-A and CS-B.

CS-A group were given an insertion sorting algorithm code developed in annotations based approach of Aspect Oriented Programming. Students typed this code segment inside an already developed project and compiled and executed. They viewed the updated code segment of the aspects visualization at the browser [AspectViz] and determined the time. Time was measured from the start of the typing of the code to view the visualization.

Similarly, CS-B group were also given a code segment consists of a method, before(), after(), pointcut and joinpoint. This segment of code was developed in the keyword-based method of Aspects Oriented Programming. Instead of a web project, this group executed the project at the eclipse and viewed the visualizations at the extended plugin of eclipse named AspectJ Development Tools [AJDT]. Likewise, all the participants of this group also determined the time. The code segments for the CS-A and CS-B groups are illustrated in below Figure 12 and 13 respectively:

```
@Before("execution(*  com.sortmeter.impl.InsertionSort.doInsertionSort(..))")
    public void logBeforeInsertionSort(JoinPoint joinPoint) {
        startTime = System.nanoTime();
        logger.debug("Method : <"+ methodName + "> started at : "+ startTime);

    }
@After("execution(*  com.sortmeter.impl.InsertionSort.doInsertionSort(..))")
    public void logAfterInsertionSort(JoinPoint joinPoint) {
        endTime = System.nanoTime();
        executionTime = endTime - startTime;
        logger.debug("Method : <"+ methodName + "> ended at : "+ endTime);
        logger.debug("Execution time <"+ methodName + "> is : "+executionTime);
    }
```

Figure 12: Code Segment for CS-A Group

```
pointcut allMethodsCut():
        execution(* (spacewar.* && !(Debug+ || InfoWin+)).*(..));

    before(): allMethodsCut() {
        if (traceMethods.getState()) {
            infoWin.println("entering " + thisJoinPoint.getSignature());
        }
    }

    after() returning : allMethodsCut() {
        if (traceMethods.getState()) {
            infoWin.println("exiting " + thisJoinPoint.getSignature());
        }
    }
```

Figure 13: Code Segment for CS-B Group

The average time of the conducted survey results in the below table:

TABLE II: AVERAGE TIME TAKEN FOR THE EXPERIMENT OF BOTH THE ASPECTVIZ AND THE AJDT

Timer Topics	Average Time for AspectViz	Average Time for AJDT
Type the code segment	6 minutes 47 seconds	6 minutes 36 seconds
Compile and run	11 seconds	10 seconds
View and find the particular visualization	23 seconds	2 minutes 8 seconds

6.0 ANALYSIS OF THE RESULTS

The first question in the first experiment is very straightforward. Since AspectViz lists all the aspects separately, it takes very little time to find the aspect's name. All the aspects details are collapsed at first. Clicking on the description link expands all the information related to that particular aspect. The main reason behind this collapsible/ expandable tabular visualization is scalability. When the number of aspects is large we need a summarized visualization which will help the developers to find a particular aspect easily. So, AspectViz can scale down and scale up the necessary aspect details as required. In AspectMaps, it also shows the aspect list separately but it doesn't show the package directly. We have to click each aspect individually in the AspectMaps window screen and see the package name from the tool-tip. So, it takes more time to answer the second part of the first question in AspectMaps and AspectViz outsmarts AspectMaps in this case.

To answer the second questions, all the pointcuts inside an aspect in AspectViz are listed in the same row where the aspect is listed. We had to go through all the rows down to the required aspect and get the pointcut details and counts. But in AspectMaps, since in package level zoom aspects are colored, it took more time, in this case, to locate the aspect and get the pointcut details.

For the third question, in AspectViz, it took almost the same time to answer the second question since this information is listed at the same location. In AspectMaps, we had to follow the same steps as in the second question and had to read all the tooltips inside that aspect and cost significant time to answer.

For the fourth question, using AspectViz it took the almost same time to answer as in the third question since answers are located in the same location for all aspects. But in AspectMaps, we had to open the "Max In" zoom option and search for the matching color aspect in the business classes and read the tooltips information to get the join point names which contribute to a greater answer time. The overall rating for AspectViz was given 9.2 and for AspectMaps was 7.95. This information represents how effective and easy AspectViz was to find the answers. So, from the point of view of usability, simplicity and effectiveness AspectViz has outsmarted AspectMaps several occasions. Below are the problems in existing tools that are effectively negotiated and resolved by AspectViz:

- Confusion with color: On several occasions, while looking for aspects with the matching color we have mistakenly located wrong aspects because of similarity of colors among multiple aspects. This led to spending more time in answering question 1. As you can see Figure 3, the two aspects GameSynchronization and DisplayAspect have almost similar colors. Since AspectViz doesn't use color, we located aspects in less time without being confused which reflects in the results.
- Complex visualization process: In AspectMaps, different shapes (rectangle, diamond, and oval) and colors are used for different AOP and class components. It takes more time to locate the targeted components (join point, advice, etc.) by coordinating colors and shapes. But, in AspectViz, all the related information of an aspect is given inside a single row which is easy to find. That's why it takes less time to answer question 4 in AspectViz than AspectMaps.
- Less visual information: AspectMaps provides different levels of visualization and to locate a join point we need to follow these levels and expand the classes using a mouse and read the tooltip information. It does not show any information (pointcut name, join point name, advice name, etc.) on the visualization window except the names of classes and aspects which are contributing more time in finding a join point. In AJDT visualizer, we have to navigate on Eclipse throughout the source code to find the targeted join point or advice. But in AspectViz, the name of the aspects, classes, pointcuts and advice are provided inside a row of tabular format, providing a higher level of visual information than most other tools.

Other problems in existing tools that cannot be measured in time also resolved in AspectViz, and discussed below:

- Dependency on third party tools: As discussed earlier, AspectMaps is an extension of moose reverse engineering tool and totally dependent on two other tools inFamix and VerveineJ to generate .mse and .xcr files respectively to visualize an AOP application. But, AspectViz is a simple web application. No need to install and directly visualize the uploaded source code without any tools needed in the middle. Hence, change in the source code is also reflected in the visualization which is not possible in AspectMaps because we have to generate .mse and .xcr files every time we change the code. Similarly, AJDT cannot work independently; it relies on Eclipse IDE to visualize the code.
- Outdated technology: The annotation feature was added in Java 5 which aids the programmers in writing less complex and clean code which is widely used in the industry. AspectJ also released all the features in annotation format along with its traditional way of writing code. AspectMaps can only visualize the AOP programs written in traditional AspectJ notations, not in the annotation. From a developer's perspective, this is a less advantageous to AspectMaps. But AspectViz can visualize programs written in AspectJ annotation to keep up with the expectations of developers.
  In the second experiment, the average time it takes to type the code segment is a bit higher for AspectViz. Compilation and runtime are almost same for both the AJDT and the AspectViz. But, the average time was taken to view and find the particular visualization section for the AspectViz is 23 seconds and for the AJDT is 2 minutes and 8 seconds. AspectViz. Compilation and runtime are almost same for both the AJDT and the AspectViz. But, the average time was taken to view and find the particular visualization section for the AspectViz is 23 seconds and for the AJDT is 2 minutes and 8 seconds.
-

  From the above test results and analysis, we can say that collapsible tabular visualization and representation of AOP programs can help the programmers to understand the flow, architecture, and behavior of applications in easier, quicker and effective way.

7.0 CONCLUSION

Here, in this paper, we have proposed new ideas and techniques to visualize AOP projects and implemented the ideas by developing a tool. The conventional way of visualizing deals with colors which can cause confusions as we discussed earlier; instead, we have used a collapsible tabular representation of aspects with one color (black) and provided the fine-grained details of each aspect in a common place. Instead of depending on other tools to analyze source code, we have directly manipulated the source code which allowed us to have more control on code and implement our logics accordingly. We have measured the performance of the AspectViz with a sample AOP project, and have compared it with the AJDT and the AspectMaps visualizer and found that the AspectViz outperformed the AJDT and the AspectMaps in many cases.

8.0 FUTURE WORK

AspectJ can be implemented in two ways: traditional keyword-based method and annotations based method. We have focused on annotations based method here only. In future, this can be extended to support the traditional keyword-based method. Our knowledge base can also be extended to search diverse AOP patterns in the program. Some frameworks also support AspectJ visualization. The same visualization can be done for those frameworks such as spring framework in Java. Moreover, better way of visualization and navigation system will always be welcome as required.

REFERENCES

[1] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An overview of AspectJ," in Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 2001), ser. Lecture Notes in Computer Science,J. L. Knudsen, Ed., no. 2072. Budapest, Hungary: Springer-Verlag, Jun. 2001, pp. 327–353.

[2] L. Ramnivas, and J Rod, Enterprise AOP with Spring Applications – AspectJ in Action, 2nd ed. Manning, 2010.

[3] A. Rountev, S. Kagan, and M. Gibas, "Static and dynamic analysis of call chains in java," in Proc. of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, ser. ISSTA '04. New York, NY, USA:ACM, 2004, pp. 1–11.

[4] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: a quantitative study. In AOSD '05: Proceedings of the 4th international conference on Aspect- oriented software development, pages 3–14, New York, NY, USA, 2005. ACM Press.

[5] J. Fabry, A. Bergel., "Design Decisions in AspectMaps", 2013 First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pp 1-4.

[6] J. Fabry, A. Kellens, S. Ducasse, "AspectMaps - A Scalable Visualization of Join Point Shadows," in 2011 IEEE 19th International Conference on Program Comprehension (ICPC), 2011, pp 121-130.

[7] F. d'Arce,E. Garcia, C. M. Correia, "Coordinated Visualization of Aspect-Oriented Programs," 2013 27th Brazilian Symposium on Software Engineering (SBES), 2013, pp 86-93.

[8] A. Colyer, A. Clement, G. Harley, and M. Webster, Eclipse aspectj: aspect-oriented programming with aspectj and the eclipse aspectj development tools. Addison-Wesley Professional, 2004.

[9] J.H. Pfeiffer and J. R. Gurd, "Visualisation-based tool support for the development of aspect-oriented programs," in AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development. New York, NY, USA: ACM, 2006, pp. 146–157.

[10] W. Coelho and G. C. Murphy, "Presenting crosscutting structure with active models," in AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development. New York, NY, USA: ACM, 2006, pp. 158–168.

[11] The Eclipse Foundation, "AJDT: Aspectj development tools," Online, 2015, http://www.eclipse.org/ajdt/ - last accessed on 12/02/2015.

[12] O.A. Lazzarini Lemos, F. Capodifoglio Zanichelli, R. Rigatto, F. Ferrari, S. Ghosh, "Visualization, Analysis, and Testing of Java and AspectJ Programs with Multi-Level System Graphs", 2013 27th Brazilian Symposium on Software Engineering (SBES), 2013, pp 49-58.

[13] O. A. L. Lemos and P. C. Masiero, "A pointcut based coverage analysis approach for aspect oriented programs," Inf. Sci., vol. 181, no. 13, pp. 2721–2746, Jul. 2011.

[14] D. Grove, G. DeFouw, J. Dean, and C. Chambers, "Call graph construction in object-oriented languages," in Proc. of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ser. OOPSLA '97. New York, NY, USA: ACM, 1997, pp. 108–124.

[15] J. Hannemann and G. Kiczales. Design pattern implementation in java and AspectJ. In Proceedings of the 17th Annual Conference on Object- oriented Programming Systems, Languages and Applications, November 2002.

[16] W. Coelho and G. C. Murphy, "Presenting crosscutting structure with active models," in AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development. New York, NY, USA: ACM, 2006, pp. 158– 168.

APPENDIX A
SAMPLE SURVEY QUESTIONNAIRES

Guidelines:

•Measure the time taken to answer each question in seconds or minutes.

•Rate each tool based on your level of satisfaction to find answers. Rate out of 10.

1.A) AspectViz: Find two Aspects and their package names?

Answer:
com.sortmeter.aspects.SelectionSortAspect.java
com.sortmeter.aspects.BubbleSortAspect.java
Time taken: 11 seconds
Rate: 10

B)AspectMaps: Find two Aspects and their package names?
Answer: DisplayAspect
Pilot
Time taken: 6 seconds
Rate: 10

2.A) AspectViz: What are the types of each pointcut in SelectionSortAspect aspect?

Answer: @Before
@After
Time taken: 7 seconds
Rate: 9
B)AspectMaps: What are the types of pointcuts in Coordinator aspect?
Answer: @After
@Before
Time taken: 14 seconds
Rate: 8
3. A) AspectViz: What are the advice (method names) inside SelectionSortAspect aspect?

Answer: logBeforeSelectionSort
logAfterSelectionSort
Time taken: 16 seconds
Rate: 9
B) AspectMaps: What are the advice (method names) inside Coordinator aspect?
Answer: synchronizationPoint
Time taken: 24 seconds
Rate: 8
4. A) AspectViz: How many Join Points does the BubbleSortAspect is operating on?

Answer: 1
Time taken: 9 seconds
Rate: 10
B) AspectMaps: How many Join Points does the DisplayAspect is operating on?
Answer: 5
Time taken: 62 seconds
Rate: 8