Evaluating Aspect Mining Techniques: A Case Study

Chanchal Kumar Roy[1], Mohammad Gias Uddin[2], Banani Roy[1] and Thomas R. Dean[2]

[1]School of Computing
Queen's University
Kingston, ON, Canada K7L 3N6
{croy, gias, broy}@cs.queensu.ca

[2]Department of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada K7L 3N6
tom.dean@queensu.ca

Abstract

Aspect mining aims at identifying cross-cutting concerns in existing systems and therefore advocates the adaption to an aspect-oriented design. This paper presents a case study examining three existing aspect mining techniques from the literature by applying them to four different open source java applications. We compare and evaluate the individual technique and confirm the findings of a previous study of combining different aspect mining techniques in order to get better results with less manual intervention.

1. Introduction

One of the major problems in legacy systems understanding, maintaining and extending is the existence of cross-cutting concerns [13]. Cross-cutting concerns are not localized in one single module but scattered over many different modules (code scattering problem), and the code of the different concerns may be tangled with the main functionality of different modules (code tangling problem)[2]. The aspect-oriented paradigm [5] provides an approach to separate such complex cross-cutting concerns in a well-modularized way, making the evolution of such systems easier and manageable. The task of detecting these cross-cutting concerns from an existing system is called *aspect mining*.

Aspect mining research assists program comprehension in two ways. First, if an aspect mining tool can find a reasonable set of concerns from a system, one can study and understand these concerns in isolation and without considering other code by following a bottom-up, divide-and-conquer comprehension strategy [4]. Second, the found concerns can be refactored to different aspects (aspect refactoring), thus improving understandability, maintainability and extensibility as well as reducing complexity of the system. Therefore, aspect mining not only helps in *aspect refactoring* but also plays an important role in program comprehension as an efficient software exploration technique [17].

While the available aspect mining techniques provide support for the process of identification of aspects in existing software, the problem is that they cannot guarantee a comprehensive aspectization of a system. This brings the idea of combining the techniques in a suitable way, and one approach towards this direction is proposed by Ceccato et al. [2, 3]. There are also other two frameworks in comparing and/or combining different techniques [14, 19]. However, the existing research describes the approach primarily on the benchmark application, JHotDraw5.4b1 [10]. In this paper, we present a case study applying three aspect mining techniques, Fan-in analysis [15], Identifier analysis [23], and Dynamic analysis [22] to four systems. We have retained the benchmark application, JhotDraw5.4b1 in our case study and added three other applications. The primary objective of this study is to evaluate the generality of the different combinations proposed by Ceccato et al. and to obtain general guidelines that can be used when applying these techniques. The major contributions of our paper can be summarized as follows:

- We confirm the major strengths and weaknesses of the three aspect mining techniques from Ceccato et al. [2], and then present the findings based on the results from the four target applications.
- We provide results of the three techniques for the target applications. This result can be used by other researchers for their own techniques, or more analysis can fine-tune our findings. In this way, it might be possible to have a common benchmark for aspect mining techniques. The results can also contribute to form a catalog of aspect-oriented refactorings [16].
- We provide an experimental analysis on the thresholds chosen for different subject programs. We also look for other existing tools and techniques in the literature.

The experience with other tools and techniques provide us some interesting lessons applicable to the three techniques used in this case study.

The remainder of this paper is organized as follows. The general methodology of this case study is discussed in Section 2, while Section 3 shows the target applications used in this study. Results obtained by the different techniques from the subject applications are provided in Section 4 whereas in Section 5, the obtained results are compared to confirm the effectiveness and weakness of the techniques used, and Section 6 summarizes our experiences learned from the study. The threats to the validity of this study are pointed out in Section 7. In Section 8, some of the works related/used in this study are briefly discussed and finally, Section 9 concludes this paper.

2. Methodology

We have used the individual methodology for each of the techniques from the literature. In the first phase, JHotDraw was used as a test bed to identify different cross-cutting concerns using the knowledge gained from the literature. The initial goal was to obtain similar results for JHotDraw to those found from literature in order to train ourselves in the used three techniques. We were able to find most of the published concerns, and we also found one/two additional potential concerns.

We have used FINT [15] for fan-in analysis and *dynamo* [22] for dynamic analysis. For identifier analysis we have developed a prototype tool in java which we call INT. We have also developed an extractor to isolate source code fragments based on the potential aspect seeds.

We applied both the top-down and bottom-up approaches of aspect mining. In the top-down approach, we focused on typical concerns that were known to be cross-cutting in the literature (e.g., persistence). We then applied the bottom-up approach. We used the techniques for an initial set of results, looked at the application's source code and documentation for a clear picture of the application to perform a manual analysis of the results. The process was repeated until we were ceased to find any new concern.

The thresholds for the fan-in analysis and the identifier analysis were determined experimentally by evaluating the concerns found for each of the thresholds. As we do not have a complete oracle, the categorization of seed and non-seed for different source codes was mainly based on the knowledge gained from existing literature.

3. Target Applications for Aspect Mining

Several open source java applications are considered in this case study. It has been noticed that some of the applications are not compatible with all the aspect mining tools. Table 1 lists the four target applications used in this study along with their statistics. The common benchmark JHotDraw is a framework for drawing structured 2D graphics and was initially developed to represent the good use of object-oriented design patterns. We have taken one application, JDraw [9] which is quite similar to JHotDraw in functionality but larger in size. It is a pixel oriented graphics editor designed especially for small to medium-sized pictures used to decorate web pages. JsokoApplet [11], a game program, is completely different from JHotDraw but similar in size. It is a java applet version of the popular sokoban game. The fourth one, SqureRootDisk [21], is a simple file and folder scanner and very small in size. Moreover, the game application is documented in German which was not easy for all of us to read through. Fortunately, one of the authors knows the German language.

4. Results of The Aspect Mining

In this section, we provide the results of applying each technique to the target applications. Due to space limitations, only partial results are provided in some cases. We have also omitted the results of JHotDraw as our found results are almost similar to the literature.

4.1. Fan-in analysis results

The results of fan-in analysis are provided in Table 2, where the number of different instances of a particular concern is indicated in the 3rd column. It is interesting to observe that some concerns, such as *consistent behavior, contract enforcement, undo,* and *persistence* are present in JHotDraw, JDraw and JSokoApplet. Moreover, the concerns have more than one instance in almost every application. We consider a functionality as cross-cutting if it is explicitly called by different methods in different different classes, where a particular method exhibits the particular functionality. Examples in this category are *consistent behavior* and *contract enforcement* in JHotDraw, JDraw, JSokoApplet, and *exception handling* in JDraw. The *path finder* functionality is the same in general throughout the JSokpApplet, although it is implemented by different methods. It is identified by looking at the similarities of the method calls and their executions. Some other concerns similar to this category are *undo, persistence* (JHotDraw, JDraw, JSokoApplet). We also find the *composite* and *observer* design patterns for JHotDraw and JDraw, however, we are unable to find any for the JSokoApplet.

The choice of threshold plays a significant role in the case of fan-in analysis while analyzing the subject programs. The results from JHotDraw do not differ much whether the chosen threshold is 10 or 5. However, for SqureRootDisk, no seed is found using thresholds 10 or

Table 1. System statistics of the subject applications used in this study

Program	Version	NCLOC	Packages	Classes	Methods	Type
JHotDraw	5.4b1	11484	17	291	2699	Drawing Editor
JDraw	1.1.5	17601	10	197	1429	Drawing Editor
JSokoApplet	1.17	10818	11	54	487	Game Applet
SquareRD	1.4.2	624	1	7	47	File, Folder Scanner

Table 2. Selection of results of the fan-in analysis

Application	Concern Type	#	Seed Description	Seed Example
JDraw	Consistent Behavior	5	Does a certain job, check or refresh the view	Tool.drawInfo(), updateTitle()
	Contract Enforcement	4	Methods perform a certain check	clipPanel.deactivate()
	Undo	1	Whether something is undoable or redoable, and does the action	DrawPixel.redo(), Undoable.-undo()
	Persistence	1	Writes bytes or long, etc.	PNGWriter.writeLong()
	Observer	3	Notifies a certain change or action performed	actionPerformed(), notifyDataListeners()
	Composite	4	Adds a new child or listener or elements	Log.addLogListener(), gui.addElement(), add()
	Exception Handling	1	Shows or debugs error messages	Log.error(), Log.debug()
JSokoApplet	Consistent Behavior	2	Refreshes the screen, checks or updates the moves or pushes	JSokoApplet.redraw()
	Contract Enforcement	1	Checks apriori conditions	Board.pushBox()
	Undo	1	Whether something is undoable or redoable, and does the action	Board.pushBoxUndo()
	Persistence	1	Saves the current levels or moves	storeBoradPosition()
	Path-Finder	4	Finds minimal path, checks path availability	calculateLowerBound()
	Prepare Game	2	Creates the graphical menu or new level	MenuBar.createMenuBar()
SquareRD	Scan	1	Scans for Files or Folders	processFile()

5. Therefore, we have experimentally reduced the threshold and observed the results for different applications. Finally, we found good results using threshold 7 for JDraw and 5 for JSokoApplet. Although JDraw is a similar type application to JHotDraw and larger in size, with threshold 10, some important concerns were missing. Similar problems occur with JSokoApplet which is quite the same size of JHotDraw. This clearly indicates that choosing a fixed threshold for all applications is not a wise decision. This was clearer when we did not find any methods from SquareRootDisk even with threshold 3 and found only one method with threshold 2.

4.2. Identifier analysis results

Using the identifier analysis technique, we found 197 concepts for JHotDraw compare to 230 concepts of Ceccato et al. using threshold 4. Using threshold 10, the number of concepts detected was significantly fewer (95) but quite similar to them (100). In both cases, 2823 elements and 419 properties were considered. A partial statistics for the remaining three applications are summarized in Table 3.

Table 3. Partial statistics of different subject programs for identifier analysis

Subject Program	#C (T=4)	#C (T=10)	#E	#P
JDraw	191	87	1626	521
JSokoApplet	90	32	510	294
SquareRD	11	2	54	49

$C = Concepts, E = Elements, P = Properties$

As noticed by Ceccato et al., the number of properties is less than the total number of elements for all the subject programs. This indicates that there are a large number of overlapping identifiers in the different source-code entities.

Table 4. Selection of results of the identifier analysis

Application	Crosscutting Concern	Concept(s)	#E	Seed Example
JDraw	Undo	undo(able)	13	addUndoable(Undoable u)
		redo	10	redo()
	Persistence and Resurrection	file(s)	13	buildLastFilesMenu()
		save(d)	23	save(), saveImage()
		write	45	writeGIF(), writeInt(i)
		read	28	readGIF(fileName), readInt()
	Observer	change(d, s)	87	dataChanged(ChangeEvent e)
		check	12	checkInput()
		listener(s)	16	notifyDataListeners(event)
	Compress Image	compress	8	compress()
	Fill Gradient	fill	15	isGradientFillOn()
	Place Text	text	15	getText(),drawText(Graphics2D)
	Composite	add	40	addColour(Color)
	Exception Handling	error	9	error(String)
		exception	7	exception(Throwable)
JSokoApplet	Persistence and Resurrection	save, saving	7	saveLevelAs(),saveLevelApplet()
		store(d)	5	storeBoardPosition(boardPosition)
		load	7	loadBackgroundGraphic(graphicname)
	Path-Finder	backward(s)	14	calculateBoxDistancesBackwards()
		forward(s)	18	calculateBoxDistancesForwards()
		reachable	12	isSquareReachable(int)
		path	12	calculatePlayerPath(int, int)
	Prepare Game	Board	46	setBoardPosition(position)
	Debugging	debug	7	printStatisticDebug()
SquareRD	Scan File	file	5	processFile(File, int)
		dir	2	getDirectorySize(File, int)
	Persistence	save	2	saveMenuItemActionPerformed

E denotes *Elements*

This is, of course, one of the premises of the identifier analysis technique.

After the initial level of stemming and filtering of identifiers, the next step is to manually analyze the seeds and decide which concepts are real seeds. As we already had experience with the subject programs from fan-in analysis and using other browsing tools, it did not take a significant time to manually analyze the candidate seeds. We have spent a total of 9 days for this part. We spent about 3 days for JHotDraw, 4 days for JDraw, 2 days for JSoko. We only needed to spend a couple of hours for SquareRootDisk. Our developed browsing/extracting tools helped us to extract the desired source code, and classify them according to the similarity of the methods/class names. Table 4 provides the final results of JDraw, JSoko and SquareRootDisk.

We used thresholds of 4, 7, 10 and 15 for all the applications, and based on the analysis of the results we took thresholds of 10 for JHotDraw, 5 for both JDraw and JSokoApplet, and 2 for SquareRootDisk.

Identifier analysis depends mainly on the naming convention. Fortunately, all the target applications used in this study follow the naming conventions. The categorization of different concepts was partially based on existing literature. However, because of the diversity of the target applications, we mainly used the documentations to define some application specific concerns, such as *path-solver* for JSokoApplet, and *compress image & paint image* for JDraw.

4.3. Dynamic analysis results

Dynamic analysis requires generation of application specific use-cases that exercise the functionalities. In the case of JHotDraw, we have used the same 27 use-cases as of Ceccato et al. For the remaining 3 subject applications, we studied the documentation of the corresponding application. In the case of JSokoApplet, we had to play the game also as the documentation was not sufficient to understand all the functionalities.

Execution of the JHotDraw use-cases listed 1962 methods, from where the concept analysis algorithm figured out 27 elements and 1962 properties. The resulting concept lattice contained 286 nodes. Even if we have used the same use-cases as of Cecceco et al. for JHotDraw, our findings are somewhat different from them. This difference in results indicates the common problem of dynamic analysis technique where the collected data can differ in each execution

as they are collected from run time executions that involves human intervention. However, the number of use-case specific and generic concepts (22 use-case specific concepts and 96 generic concepts) is similar to them. In order to remove false positives, we have manually filtered out some concern seeds and grouped similar concern concepts. The number of use-cases (elements), properties (methods) and the nodes of the remaining three subject programs found from our analysis are summarized in Table 5.

Table 5. Statistics of different subject programs for dynamic analysis

Application	Use-cases	Properties	Nodes
JDraw	33	664	594
JSokoApplet	20	239	70
SquareRD	4	18	4

Among the concepts in the lattice, some of them have satisfied the cross-cutting conditions (scattering and tangling) for the use-case specific concepts and some have satisfied the conditions for the generic concepts. Then both the use-case specific and generic concepts were revisited manually in order to determine the real seeds and to avoid false positives.

The list of the candidate concerns for JDraw, JSokoApplet and SquareRootDisk are summarized in Table 6. In

Table 6. Selected results of dynamic analysis

Application	Crosscutting Concern	#C	#M
JDraw	Resize	2	21
	Observer	5	15
	Reset Alpha Values	1	7
	Compress Image	2	16
	Persistence	2	22
	Insert Image in Frame	1	8
	Create and Fill Gradient	2	4
	Remove Color From Local Palette	1	25
	Swap Color In Local Palette	1	7
	Undo	4	19
JSokoApplet	Path-Finder	4	31
	Deadlock Detection	1	9
	Undo	1	4
	Prepare Game	1	8
SquareRD	Scan	1	12
	Persistence	1	5

C denotes $Concepts$, and **M** denotes $Methods$

the initial assessment of JDraw, 28 use-case specific and 97 generic concepts were identified and finally, 35 concepts were judged to be associated with 24 cross-cutting concerns. In the initial assessment of JSokoApplet, 3 use-case specific and 17 generic concepts were identified and finally, 9 concepts were judged to be associated with 6 cross-cutting concerns. In the initial assessment of SqureRoot-Disk, 2 use-case specific and 2 generic concepts were identified and finally, 2 concepts were judged to be associated with 2 cross-cutting concerns. For the case of JHotDraw, 36 concepts were judged to be associated with 16 cross-cutting concerns. The methods associated with each candidate seed indicate the aspectizable functionality of that particular seed. As dynamic analysis is partial, the results may not be complete and may have some false positives also.

5. Observing and Comparing the Results

In this section, we discuss some selected concerns identified by the different techniques from the target applications.

5.1. Concerns found by the techniques

The list of concerns identified by the different techniques for JDraw (partial result), JSokoApplet and SquareRoot-Disk are summarized in Tables 7, 8 and 9 respectively. Here, the first column represents the name of the concern, the other columns show by which technique(s) the concern was identified. The + sign indicates that the specific concern was identified by the corresponding technique.

Table 7. Detected concerns in JDraw

Crosscutting Concern	Fan	Iden	Dyn
Consistent Behavior	+	-	-
Contract Enforcement	+	-	-
Undo	+	+	+
Persistence	+	+	+
Observer	+	+	+
Composite	+	+	-
Exception Handling	+	+	-
Drawing Figure	-	+	+
Compress Image	-	+	+
Resize	-	+	+
Paint Image	-	+	+
Manage GUI	-	-	+
Zoom Image	-	-	+
View Animation	-	-	+

It is observed from the tables that none of the techniques are self-sufficient to discover all the concerns from an application. The percentage of concern coverage found by each of the techniques is noted in Table 10, where the second column represents the total number of concerns found by all the techniques. Considering this number as the total number of concerns for an application, we have calculated the percentage of concern coverage for a particular technique. Here it is found that the average percentage cover-

Table 8. Detected concerns in JSokoApplet

Crosscutting Concern	Fan	Iden	Dyn
Consistent Behavior	+	-	-
Contract Enforcement	+	-	-
Undo	+	-	+
Persistence	+	+	-
Path-Finder	+	+	+
Composite	+	-	-
Prepare Game	+	+	+
Debug	+	+	-
Transform Graph	-	+	+
Deadlock Detection	-	+	+
Set Editor Mode	-	+	+
Path-Solver	-	+	-

Table 9. Detected concerns in SquareRD

Concern	Fan	Iden	Dyn
Scan	+	+	+
Persistence	-	+	+

age is the highest (73.9%) using dynamic analysis experiment. Although dynamic analysis is partial, this technique can provide more concern seeds than the others. Even if it has missed some major concerns, it still provides promising results. On the other hand, for fan-in analysis technique, the result depends on the threshold chosen. The results of the identifier analysis is also based on threshold and it might differ depending on the threshold used. Therefore, the percentage of concern coverage found by different techniques might significantly differ depending on the target applications used, as well as on the particular characteristics specific to the particular techniques such as the use-cases for dynamic analysis and the threshold for fan-in and identifier analysis techniques. Although different techniques can

Table 10. Percentage of concerns found by different techniques

Application	Total	Fan	Iden	Dyn
JHotDraw	29	55.17%	34.48%	58.62%
JDraw	31	25.8%	27%	87%
JSokoApplet	12	66.6%	58.33%	50%
SquareRD	2	50%	100%	100%
Average	-	49.39%	54.95%	73.9%

have different levels of concern coverage, it might be interesting to see how efficient a technique is in providing information for a particular concern. If we have a quick look at the results of different techniques, we will see that identifier analysis technique discovers more methods/classes than the other two techniques which indicates that identifier analysis technique might be able to provide more information for a particular concern. As this technique gives many false positives, we obtain the percentage of false positives for some particular concerns. We use the common concerns discovered by all the techniques for each application.

Table 11 shows the percentage of false positives of the 6 concerns *Undo, Persistence, Observer, Path-finder, Prepare Game and File & folder scanner* taken from different applications. The data indicates that identifier analysis technique gives more false positives than the other two technique while fan-in analysis technique gives fewer false positives or even accurate results. However, as identifer analysis and dynamic analysis techniques discover more concern elements, it is worth while to see how complete a technique is for providing information about a particular concern. For having such statistics, we have considered the same 6 common concerns. First of all, we have gathered all the methods/classes returned by all the techniques related to a particular concern and then filtered out the false positives keeping the real elements for that concern. We have considered this number of methods/classes of a concern as a complete concern information and calculated how efficient the different techniques are in covering a complete concern. In Table 12 , we have provided the statistics considering the previous 6 common concerns. In this table, we see that even through identifier analysis returns more false positives than the others, it has more percentage coverage for a concern. On the other hand, fan-in analysis can only cover a small fraction of a complete concern. Although dynamic analysis is partial, it can still provide good concern coverage, and gives less false positives than identifier analysis.

5.2. Complementarity

Fan-in analysis detects methods specific to a concern which are scattered all over the subject program. It tends to miss the methods which are not scattered that much in the program, and thus it misses methods of low fan-in value. Dynamic analysis tends to detect methods specific to particular execution trace. It tends to miss the methods which are likely to occur in all execution traces. Identifier analysis relies on specific naming conventions and tends to produce detailed results covering most of the methods detected by both the fan-in and dynamic analysis techniques. This detailed result might not always helpful while considering a large application. In this case, the manual analysis part is highly rigorous and time-consuming. This disadvantage leads to the conclusion that identifier analysis is not helpful for larger subject programs. Table 13 gives an interesting observation while combining the different results from the three techniques. From the statistics, it is clear that most of the concerns can be identified and found by an effective combination of the results obtained from fan-in analysis technique and dynamic analysis technique. We have found that identifier analysis fails to identify any new concern in

Table 11. Percentage of false positives for selected concerns by different techniques

Concern	JHotDraw			JDraw		
	Fan	Iden	Dyn	Fan	Iden	Dyn
Undo	27%	50%	35%	12%	43%	9%
Persistence	20%	21%	19%	0%	35%	50%
Observer	0%	84%	16%	20%	65%	31%
Average	16%	52%	23%	11%	48%	30%
Concern	**JSokoApplet**			**SquareRD**		
	Fan	Iden	Dyn	Fan	Iden	Dyn
Path-Finder	17%	50%	27%	-	-	-
Prepare Game	0%	61%	0%	-	-	-
File & Folder Scanner	-	-	-	0%	43%	29%
Average	9%	56%	9%	0%	43%	29%

Table 12. Percentage of concern coverage for selected concerns by different techniques.

Concern	JHotDraw			JDraw		
	Fan	Iden	Dyn	Fan	Iden	Dyn
Undo	10%	94%	34%	32%	59%	86%
Persistence	20%	100%	83%	5%	82%	24%
Observer	65%	81%	48%	8%	83%	31%
Average	32%	92%	55%	15%	75%	41%
Concern	**JSokoApplet**			**SquareRD**		
	Fan	Iden	Dyn	Fan	Iden	Dyn
Path-Finder	14%	48%	41%	-	-	-
Prepare Game	15%	67%	67%	-	-	-
File & Folder Scanner	-	-	-	12%	23%	65%
Average	15%	58%	54%	12%	23%	65%

Table 13. Concerns identified by either fan-in or dynamic analysis

Application	Total	Fan	Dyn	Fan ∪ Dyn	Fan ∩ Dyn	Total-(Fan ∪ Dyn)
JHotDraw	29	17	16	29	4	0
JDraw	31	8	26	30	4	1
JSokoApplet	12	8	6	11	2	1
SquareRD	2	1	2	2	1	0

case of JHotDraw compared to the other two techniques. However, identifier analysis did manage to find two specific concerns in the case of JSokoApplet (*Path-solver*) and JDraw (*Paint-image*). *Path-solver* gives the solution according to a predefined search strategy. It is little bit different compared to *path-finder* which checks for the existence of a path. *Path-solver* was found because of the specific naming conventions used for it in JSokoApplet. This concern could not be differentiated in the dynamic analysis because of the lack of a suitable use-case specific to it. Fan-in analysis also misses this concern because of the low fan-in values of the associated methods. For JDraw *Paint-image* is identified using the keyword *paint*. Based on the existing result we can say that we can even avoid identifier analysis technique in the first phase of aspect mining. However, for a complete coverage and to have more percentage coverage of a concern, we do really need identifier analysis technique.

6. Lessons Learned

While mining aspects from the 4 different systems several interesting observations were noticed. For JHotDraw, it was fairly easy to find out the concerns as they have already been discovered and the methods belonging to those concerns are well-known. It was not the same for the case of an unfamiliar application where we were not sure what could be the possible aspect candidates. It became clear that the analysis of the initial results from the techniques requires domain knowledge to accomplish the task with any accuracy. Only a small portion of the results can be discovered without reading the documentation. Documentation was also essential to make the use-cases for dynamic analysis and to analyze different names for identifier analysis.

Analysis of JDraw using the documentation available online was fairly easy. Also, the similar nature of JDraw com-

pared to JHotDraw helped us to mine the aspects. Interesting observations were made when we started working with the interactive game application, JSokoApplet, where the documentation is actually the comments and those are incomplete. This inability to have a complete documentation created a significant problem while analyzing the JSokoApplet. We opted for two approaches to solve the problem. In the first phase, we became familiar with the game by playing it exhaustively and analyzing the moves. In the second phase, we looked at the documentation available in the source code comments to gather a clear idea on the particular behavior of different classes and methods. These two approaches helped us overcoming the unavailability of a complete documentation for JSokoApplet.

SquareRootDisk is a very small program, and because of its simplicity and small size, it was not hard for us to capture the behavior of the application. Therefore, making use-cases and performing dynamic analysis were simple.

The results clearly indicate that choosing a fixed threshold for all applications is not reasonable. This was evident when we did not find any methods from SquareRootDisk even with threshold 3. From our observations it was noticed that the decision of choosing the threshold value depends mainly on the number of methods of the target application. The correlation between the found concerns and the number of methods of the target applications is better than other system properties such as lines of code (LOC) or number of packages. There is also a good correlation of the found concerns with the number of classes.

Our data shows that a good initial threshold for fan-in analysis can be given by $Threshold = 0.0028 * No.\ of\ Methods + 2.7$ with a correlation coefficient of 0.97. Taking number of classes into consideration, we get $Threshold = 0.025 * No.\ of\ Classes + 2.5$ with correlation coefficient of 0.97. The average of the thresholds obtained from the above two linear equations is close to the determined experimental thresholds. The results are preliminary as we have used only 4 data points but it suggests that the number of methods and classes can be used to provide an initial starting point for the analysis.

We applied similar approach for the thresholds of identifier analysis. In this case, we have found a linear relationship between the threshold and the sum of the number of methods and classes of the target application. The linear equation, $Threshold = 0.0024 * (No.\ of\ Methods + No.\ of\ Classes) + 2.4$ with correlation coefficient 0.94 is a good predictor for an initial threshold.

Interesting observations were noticed in the case of dynamic analysis experiments also. The results of an application using this technique heavily rely on the use-cases that exercise the functionalities of that application. Even the same set of use-cases can produce different set of results from the same application. It is also challenging for some applications such as interactive game application (e.g., JSokoApplet) to derive all the use-cases even if a reasonable documentation is available.

While it was easy to identify some known design pattern specific concerns (e.g., *composite* and *observer*), we are still not sure whether other design patterns can be identified as easily, especially when we cannot be sure about their implementation in the source code.

We have noticed that applying the identifier analysis techniques on the results of Timna [19] would speed up the process by reducing manual efforts. Timna outputs a set of candidate methods from an application. Identifier analysis technique is then applied to those methods and their associated class names only, instead of applying to all the methods and class names of the application. Other combinations like the results of dynamic analysis for a particular concern can be matched against the results obtained from Timna and identifier analysis or the union results of fan-in analysis and dynamic analysis can be used for identifier analysis as in Ceccato et al., but only will be applicable to the results of Timna instead of applying to the whole application.

As in Shepherd et al. [19], it has been observed that exploration tools are effective and useful when used in conjunction with a seed identifier. In this study, we have used FEAT [18], JQuery [24], Prism [25], AMT [7], Aspect Browser [6] and CCFinder (clone detection tool) [12]. In our experience, these tools are not that useful by themselves for aspect mining. We then used the tools after obtaining the seed identifier from the techniques and obtained better results with less manual works.

We have also investigated the possibility of using clone detection techniques for the subject programs of our study. Zhang et al. [26] and Bruntink et al. [1] explore this technique in novel directions. Our experience is that clone detection technique may assist both the fan-in analysis and identifier analysis techniques. For fan-in analysis technique, clone detection technique might increase the fan-in value of a method that is duplicated somewhere else. In the case of identifier analysis, clone detection might increase the percentage of concern coverage and at the same time might assist removing the false positives by looking for similar code clones in the target applications while expanding the seed for a concern. It is also felt that a semantic clone detection technique could assist aspect mining process as it could identify code clones based on semantic similarity.

7. Threats to the Validity

One of the major threats to the results of this study is the lack of a sound definition of cross-cutting concerns. Moreover, the heterogeneity in the search-goals of the considered techniques bounds the comparison criteria applicable

to only a selected common findings. An approach overcoming these difficulties is proposed by Marin et al. [14].

Although we have presented a linear relationship between the number of methods (or classes or a combination of those) of the target application to the initial threshold for fan-in analysis and identifier analysis, it is still not clear whether these linear relationships are applicable to very large systems. Again, this uncertainty raises the question of having a proper definition of cross-cutting concerns w.r.t the maximum/minimum allowable threshold values of fan-in analysis and identifier analysis techniques.

8. Related Works

This work is directly related to the work of Ceccato et al. [2, 3] that provides several interesting combinations of the fan-in analysis, identifier analysis and dynamic analysis and applies these combined techniques to JHotDraw.

Marin et al. [14] provide a novel approach of comparing different techniques in a search-goal oriented way based on cross-cutting concern sorts. As our experiment was conducted prior to the publication of this paper, we have not considered their approach in our study.

Shepherd et al. [19] propose a combined framework called Timna of several aspect mining techniques. In Timna, a kind of meta approach has been used which allows one to evaluate several mining approaches. Machine learning techniques are used to combine the aspect mining techniques in an automated way. However, annotation of cross-cutting concerns on some training application is required in this framework.

Bruntink et al. [1] present a case study that evaluates clone detection techniques for identifying cross-cutting concerns. However, the domain of the study limits its applicability.

Lexical search-based tools, such as the Aspect Mining Tool (AMT) [7] and the Aspect Browser [6] are designed to leverage the power of a lexical search. The Aspect Browser provides text-based mining which is basically a string pattern-matching technique to discover aspects. One can specify a regular expression that describes the code belonging to the aspect of interest and a color. The tool then identifies the code conforming to the regular expression and highlights it using the associated color in the source code editor.

AMT is an extension of the Aspect Browser which combines the text-based and type-based mining, and considers types in identifying cross-cutting concerns. By taking into account the type information, it ensures fewer false positives and false negatives.

The Prism tool [25] extends the AMT by providing *type ranking* feature and taking into account control flow information. This tool assumes that types that are used widely in the application are a good indicator of cross-cutting code, and it ranks the types in the system according to their use.

Exploratory tools are less automatic than lexical search tools. Starting with a seed, a user of such tools can navigate a subject application via structural queries. These navigation tools, like JQuery [24] and FEAT [18] help the user in navigating the application, but the user has to take her own conclusions about the code.

Harman et al. [8] provide a slicing based aspect mining technique. The developer points out a particular expression or statement and a tool automatically computes the corresponding slice. The code segment computed in this way can then be refactored into an aspect.

Based on the combination of a program dependence graph (PDG) and abstract syntax tree (AST), Shepherd et al. [20] provide a fully automatic aspect mining and refactoring tool, Ophir. The identification algorithm starts only at specific points of each method in order to speed up the processing time.

9. Conclusions

Because of the diverse nature of different aspect mining techniques, it is still not clear how to combine the different techniques in order to obtain a comprehensive result. However, Ceccato et al. propose an approach to this direction which motivated us to do this case study. We have verified and confirmed the findings of Ceccato et al. using four different subject programs. The findings from this case study may assist in obtaining a standard combination of the existing techniques instead of relying on a specific one.

While working on this case study, the necessity of a suitable aspect mining tool was felt that could combine the strengths of different techniques while avoiding their limitations. We also faced difficulty in analyzing the source code as considerable manual effort was required. The results presented in this case study can be refined further to get a better result and to have more fine-tuned statistics for percentage false positives and percentage concern coverage.

Future work mainly focuses on building a better aspect mining tool combining available aspect mining techniques. As said, significant amount of manual works are required in the aspect mining process. Thus, another further work may focus on getting benefits from the program comprehension techniques in the manual analysis part of aspect mining.

10. Acknowledgements

The authors would like to thank Mariano Ceccato, David Shepherd and Marius Marin for their help on providing necessary data, useful comments and suggestions. The authors also thank the three anonymous reviewers for their valuable comments and suggestions in improving the paper.

References

[1] M. Bruntink, A. van Deursen, R. van Engelen and T. Tourwe. On the Use of Clone Detection for Identifying Crosscutting Concern Code. In *IEEE Trans. Software Eng*, 31(10): 804-818, 2005.

[2] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, T. Tourwé. Applying and combining three different aspect Mining Techniques. In *Software Quality Journal*, 14(3): 209-231, Sep. 2006.

[3] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, T. Tourwé. A Qualitative Comparison of Three Aspect Mining Techniques. In *Proceedings of the 13th International Workshop on Program Comprehension*, pp. 13-22, Missouri, USA, May 2005.

[4] A. Deursen, M. Marin, and L. Moonen. Aspect mining and refactoring. In *Proc. of the First Int. Workshop on REFactoring: Achievements, Challenges, Effects*, Nov. 2003.

[5] R. E. Filman, T. Elrad, S. Clarke and M. Aksit. *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[6] W. G. Griswold, Y. Katoy and J. J. Yuan. Aspect Browser: Tool Support for Managing Dispersed Aspects. In *First Workshop on Multi-Dimensional Separation of Concerns*, OOPSLA, Denver, Nov. 1999.

[7] J. Hannemann, and G. Kiczales. Overcoming the Prevalent Decomposition in Legacy Code. In *Workshop on Advanced Separation of Concerns*, ICSE, Toronto, May 2001.

[8] M. Harman, L. Hu, M. Munro, X. Zhang, D. W. Binkley, S. Danicic, M. Daoudi, and L. Ouarbya. Syntax-directed amorphous slicing. In *Journal of Automated Software Engineering*,11(1):27-61, Jan. 2004.

[9] JDrawV1.1.5. `http://www.j-domain.de/` (June 2006)

[10] The JHotDraw v5.4b1 Drawing Tool. `http://www.jhotdraw.org/` (June 2006)

[11] JSokoAppletV1.17. `http://sourceforge.net/projects/jsokoapplet` (June 2006)

[12] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. In *IEEE Trans. Software Eng.*, 28(7):645-670, Jul. 2002.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier and J. Irwin. Aspect-Oriented Programming. In *Proc. of the European Conference on Object-Oriented Programming*, Springer-Verlag LNCS 1241, Jun. 1997.

[14] Marius Marin, Leon Moonen, Arie van Deursen. A common framework for aspect mining based on crosscutting concern sorts. In *Proc. of the 13th Working Conference on Reverse Engineering*, pp.29-38, Benevento, Oct. 2006.

[15] M. Marin, A. V Deursen, and L. Moonen. Identifying Aspects using Fan-in Analysis. In *Proc. of the 11th Working Conference on Reverse Engineering*, pp. 132-141, DUT, Nov. 2004.

[16] M. P. Monteiro and J. M. Fernandes. Towards a catalog of aspect-oriented refactorings. In *Proc. of the 4th Intl. Conference on Aspect-Oriented Software Development*, pp. 111-122, Chicago, Mar. 2005.

[17] L. Moonen. Exploring software systems. In *Proceedings of the International Conference on Software Maintenance*, Amsterdam, Sep. 2003.

[18] M. P. Robillard and G. C. Murphy. Concern Graphs: Finding and Describing Concerns using Structural Program Dependencies. In *Proc. of the 24th Intl. Conference on Software Engineering*, pp. 406-416, Orlando, May 2002.

[19] D. Shepherd, J. Palm, L. Pollock and M. Chu-Carroll. Timna: A Framework for Automatically Combining Aspect Mining Analyses. *Intl. Conference on Automated Software Engineering*, pp. 184-193, 2005.

[20] D. Shepherd, E. Gibson, E. and L. Pollock. Design and Evaluation of an Automated Aspect Mining Tool. In *Intl. Conference on Software Engineering Research and Practice*, pp. 601-607, Las Vegas, Jun. 2004.

[21] SquareRDV1.4.2 `http://sourceforge.net/projects/squarerootdisk/`(June 2006).

[22] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In *Proc. of the 11th Working conference on Reverse Engineering*, pp. 112-121, DUT, Nov. 2004.

[23] T. Tourwé and K. Mens. Mining Aspectual Views using Formal Concept Analysis. In *Proc. of the 4th IEEE Intl. Workshop on Source Code Analysis and Manipulation*, pp. 97-106, Chicago, Sep. 2004.

[24] K. D. Volder. JQuery: A Generic Code Browser with a Declarative Configuration Language. In *Proc. of PADL*, Charleston, Jan. 2006.

[25] C. Zhang and H.-A. Jacobsen. PRISM is Research In aSpect Mining. Software Demonstration at *OOPSLA'04*, Vancouver, Oct. 2004.

[26] J. Zhang, J. Gray, Y. Lin, and R. Tairas. Aspect Mining from a Modeling Perspective. In *Intl. Journal of Computer Applications in Technology*, 00(0/0), 2006.