Exploring AspectJ Refactorings

Geeta Bagade(Mete)
Ph.D. Scholar,
YMC, Bharati Vidyapeeth,
Pune, India

Shashank Joshi, PhD
Professor/ Ph.D Guide,
COE, Bharati Vidyapeeth,
Pune, India

ABSTRACT
Refactoring is nothing but a change that you make to the software. It is a series of steps that are carried out on the piece of software. After the refactoring is applied on the code it is important to note down the changes that have been done to the software. Care should be taken such that the behvaiour of the software does not change even if the refactoring is applied but its execution time, performance increases. This paper is in continuation with the other refactorings that have been already presented. Here we present three more refactorings that have been identified. The refactorings are applied on the projects and the results are compared before the refactoring is applied and after the refactoring is applied.

General Terms
Aspect Oriented Programming, AspectJ.

Keywords
Refactoring, AspectJ, abstract, extends, pointcut, Inheritance, aspect, Joinpoint, crosscutting concern, Refactoring Mechanism

1. INTRODUCTION
Refactoring is a process where the code in the software is changed. A simple refactoring could be changing the name of the variable. Refactoring is generally applied when you want to make some changes to the software. The process of refactoring can be done manually or it can be automated. There are various techniques that are used for refactoring. Some of them are Graph Transformations, Program Refinement, Formal concept Analysis, Assertions, Software Metrics and Program Slicing. Each of this technique guarantees that the behaviour of the software is preserved after the refactoring is applied to the software. The process of refactoring consists of various steps. The process starts with identification of the place in the code where the refactoring should be applied. One of the ways of identifying the place where the refactoring should be applied is a "bad smell" in the software. In this paper we present three more refactorings. Each of the refactoring will be explained in terms of what it is; mechanism used to apply it, comparison of the code before and after the refactoring is applied. The systems that have been used and the software metrics that is used for calculation is already explained in [3]

2. NEW REFACTORINGS IDENTIFIED

2.1 Name of the Refactoring: Remove Aspect Inheritance (One Aspect Extends Another Aspect)
An aspect that is declared as "abstract" can be extended by another aspect. A "concrete" aspect cannot be

inherited. An abstract aspect can declare abstract methods and abstract pointcuts. The inheriting aspect should provide the code for the abstract methods and abstract pointcut.

Refactoring Mechanics:
1. Select the aspect that is to be refactored
2. If the abstract aspect contains static fields, then simply prefix the names of the fields with the name of the aspect else go to step3 E.g.: AspectName.FieldName
3. Create an Inner aspect inside the aspect which is the "sub aspect"
4. Declare the Inner aspect as static
5. Move the methods and the pointcuts from the "sub aspect" to the inner aspect
6. Create object of the newly created "inner aspect" inside the "sub aspect" which is now an "outer aspect"
7. In the "outer aspect" provide appropriate references to the methods of the "inner aspect"
8. Leave the "super aspect" as it is
9. Test the refactoring after each method is moved and references are provided

2.2 Name of the Refactoring: MAKE Abstract Pointcut as Non Abstract in Abstract Aspect
We can declare an aspect as abstract if it has abstract method or abstract pointcut. The word "abstract" is prefixed before the name of the method or the name of the pointcut. A pointcut that is declared as abstract should end with a semicolon and should not have any body of code. The abstract pointcuts also have advices defined on them. The abstract pointcut is made concrete in the sub aspect in which it is inherited. Here we are making the abstract pointcut non abstract. So the concrete aspect will have two pointcuts : one that is already present within itself and the other which it has inherited from the abstract aspect.

Refactoring Mechanics 1*:*
1. Identify the pointcut that should be made non abstract
2. Delete the word "abstract" from the pointcut declaration
3. Provide appropriate code or do nothing code for the pointcut inside the aspect.
4. Test the code that has been refactored for behaviour preservation

Note: If the abstract pointcut is deleted from the abstract aspect, errors are introduced in the aspect. So we have another mechanics as described under

Refactoring Mechanics 2

1. Identify the pointcut that should be made non abstract
2. Delete the "abstract" pointcut
3. Move the advices code from the abstract aspect to the concrete aspect appropriately
4. Test the code that has been refactored for behaviour preservation

2.3 Name of the Refactoring: Remove The "Extends" Keyword From The Aspect Declaration

One concrete aspect cannot inherit another concrete aspect in AspectJ. It can only extend an abstract aspect. In case of a concrete aspect that extends another aspect which is abstract, the keyword "extends" tells the AspectJ compiler that the concrete aspect should provide the definition for each abstract method and abstract pointcut. Also if the aspect is extending another aspect that the advices that are written in the subaspect get higher precedence as compared to the advices in the superaspect. So in this case the subaspect overrides the behaviour of the superaspect.

Refactoring Mechanics

1. Identify the aspect for which you want to remove the "extends" keyword
2. Remove the "extends" keyword from the subaspect
3. Move the constructor code from the "super aspect" to the "sub aspect"
4. Copy the code that is written for the advices from the superaspect to the subaspect appropriately
5. Delete the aspect if required
6. Test whether the code preserves the behaviour

3. RESULTS OF THE RESEARCH

3.1 Name of the Refactoring:Remove Aspect Inheritance (One Aspect Extends Another Aspect)

No.	Para	FS		SOP		SW		TE	
		Before	After	Before	After	Before	After	Before	After
1.	VS	14	14	7	7	21	21	6	6
2.	NA	11	12	6	9	50	55	1	2
3.	NO	31	32	19	26	108	131	22	37
4.	WOC	67	68	29	40	196	252	42	52
5.	NS	53	55	32	47	460	570	51	64
6.	LC	199	201	102	132	1472	1678	119	167
7.	DIT	15	15	10	10	29	28	5	5
8.	NC	0	0	1	1	4	4	2	2
9.	TE	40.33	37	329.33	321	23932	18491	288	261

Fig. 1. Refactorng 2.1 Comparison Table

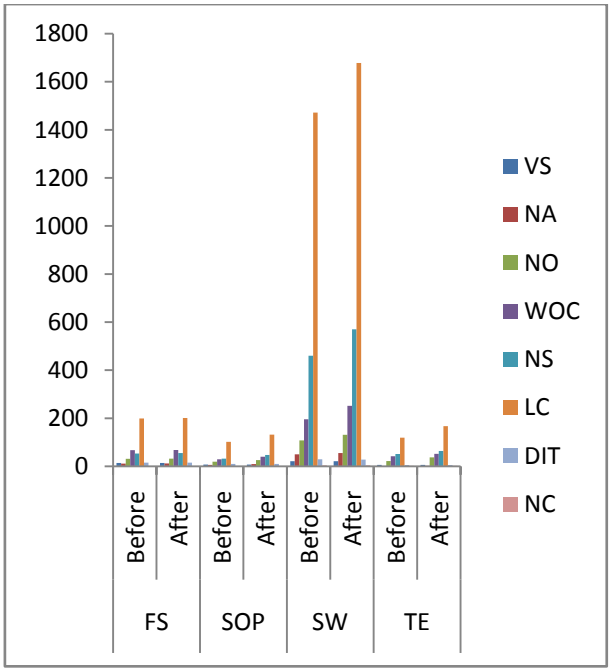

Fig. 2. Refactorng 2.1 Comparison Chart

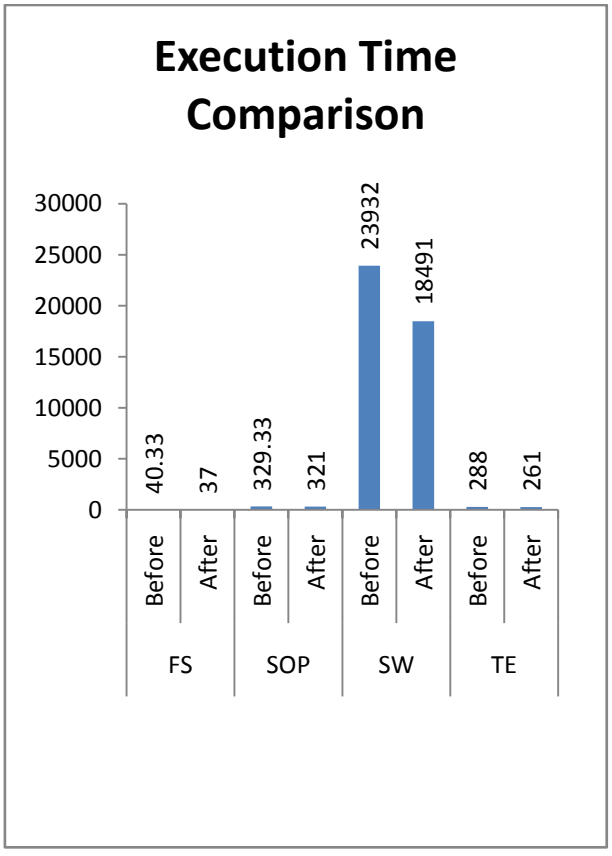

Fig. 3. Refactorng 2.1 Execution Tme Comparison Chart

3.2 Name of the Refactoring: Make Abstract Pointcut As Non Abstract In Abstract Aspect

No.	Para	SOP		SW	
		Before	After	Before	After
1.	VS	7	7	21	21
2.	NA	6	6	50	50
3.	NO	19	19	108	108
4.	WOC	29	29	196	196
5.	NS	32	32	460	460
6.	LC	102	102	1472	1472
7.	DIT	10	10	29	29
8.	NC	1	1	4	4
9.	TE	329.33	321	23932	10533

Fig. 4. Refactorng 2.2 Comparison Table

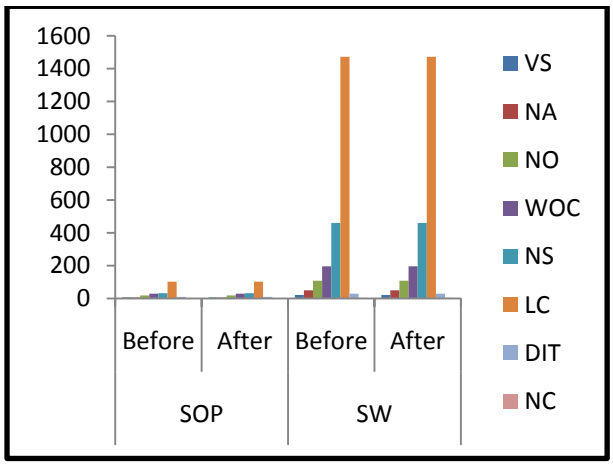

Fig. 5. Refactorng 2.2 Comparison Chart

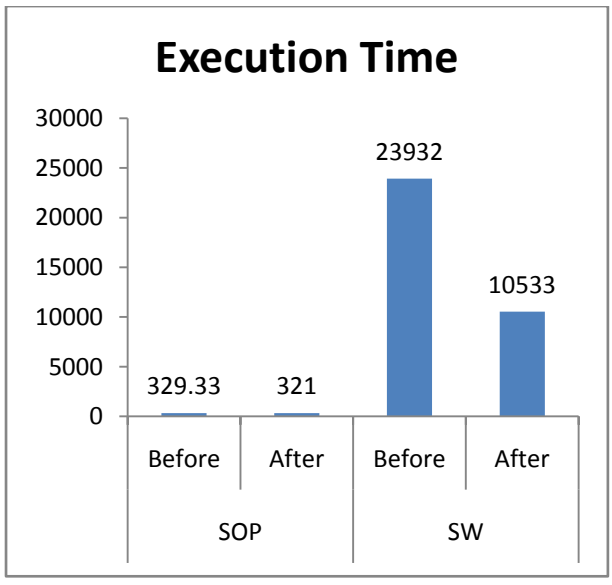

Fig. 6. Refactorng 2.2 Execution Tme Comparison Chart

3.3. Name of the Refactoring: Remove The "Extends" Keyword From The Aspect Declaration

No.	Para	FS		SOP		SW		TE	
		Before	After	Before	After	Before	After	Before	After
1.	VS	14	13	7	6	21	20	6	5
2.	NA	11	15	6	6	50	50	1	1
3.	NO	31	31	19	19	108	107	22	36
4.	WOC	67	67	29	29	196	195	42	72
5.	NS	53	54	32	35	460	460	51	64
6.	LC	199	196	102	100	1472	1474	119	162
7.	DIT	15	13	10	11	29	27	5	5
8.	NC	3	0	1	0	4	4	2	0
9.	TE	40.33	36	329.33	321	23932	21010	288	144

Fig. 7. Refactorng 2.3 Comparison Table

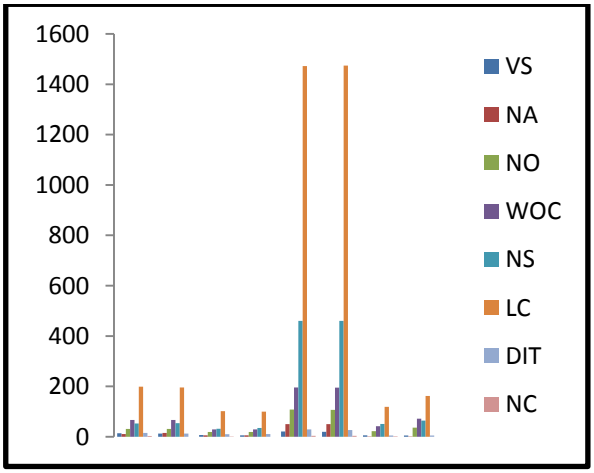

Fig. 8. Refactorng 2.3 Comparison Chart

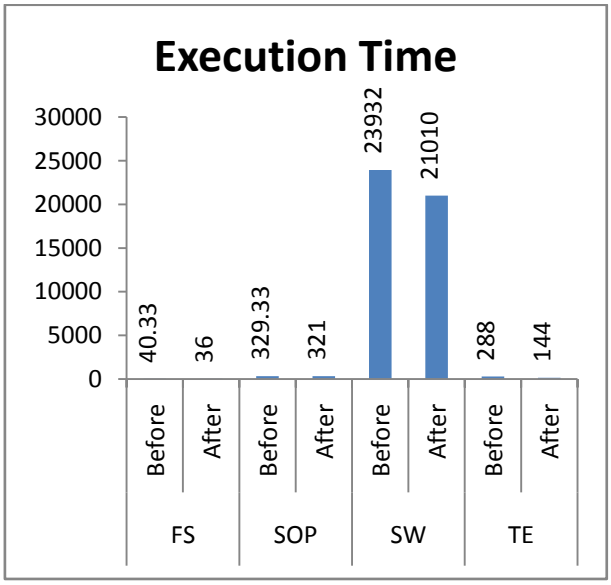

Fig. 9. Refactorng 2.3 Execution Tme Comparison Chart

4. CONCLUSION

As seen in refactoring 2.1, the values for vocabulary size and number of children shows no change, but other parameters like number of attributes, number of operations, weighted operations per component, lines of code, depth of inheritance

tree either show an increase or decrease in their values. However the parameter execution time shows a decrease after the refactoring is applied. In case of refactoring 2.2, all the parameters remain same except execution time. The execution time has decreased after the refactoring is applied. In case of refactoring 2.3, all the parameters show an increase or decrease in the values. But again the execution time has reduced after the proposed refactoring is applied. This is true in case of all refactorings. It can therefore be concluded that the applied refactorings will help the code in executing faster and therefore give good performance.

5. REFERENCES

[1] Adams B, Meuter W.D., Tromp H, Hassan A. "Can we refactor conditional compilation into Aspects?" Proceedings of the 8th ACM international conference on Aspect-oriented software development. ACM New York, NY, USA, 2009. 243-254.

[2] Geeta B, Shashank J, More refactorings for Aspect Oriented Systems using AspectJ, International Journal of Innovative Research in Computer and Communication Engineering ,(An ISO 3297: 2007 Certified Organization), Vol. 4, Issue 4, April 2016

[3] Geeta B, Shashank J, Analysis of Aspect Oriented Systems: Refactorings using AspectJ, International Journal of Computer Sciences and Engineering

[4] Geeta B, Shashank J, Some thoughts on Refactoring for Aspect Oriented Programming using AspectJ, International Journal of Scientific & Engineering Research, Volume 5, Issue 6, June-2014 , pp 561-564

[5] Gustavo Soares, Melina Mongiovi, and Rohit Gheyi. "Identifying Overly Strong Conditions in Refactoring Implementations,." 27th IEEE Internationall Conference on Software Maintenance. 2011.

[6] Fabiano C. Ferrari, Bruno B. P. Cafeo, Thiago G. Levin, Jésus T. S. Lacerda, Otávio A. L. Lemos, José C. Maldonado and Paulo C. Masiero. "Testing of aspect-oriented programs: difficulties and lessons learned based on theoretical and practical experience." Journal of the Brazilian Computer Society (2015): DOI: 10.1186/s13173-015-0040-1.

[7] Khine Zar Ne Winn. "Quantifying and Validation of Changeability and Extensibility for Aspect-Oriented Software." International Conference on Advances in Engineering and Technology (ICAET'2014). Singapore, March 29-30,2014 .

[8] Melina Mongiovia, Rohit Gheyia,Gustavo Soares,Leopoldo Teixeirab,Paulo Borba. "Making Refactoring Safer through Impact Analysis." Science of Computer Programming. 2013.

[9] Michael Mortensen, Sudipto Ghosh. "Aspect-Oriented Refactoring of Legacy Applications: An Evaluation." IEEE Transactions on Software Engineering . 2012.

[10] M. Badri, A. Kout and L. Badri. "On the effect of aspect-oriented refactoring on testability of classes: A case study." Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on, Sharjah. doi: 10.1109/ICCSII.2012.6454577, 2012. 1-7.

[11] Melina Mongiovia, Rohit Gheyia,Gustavo Soares,Leopoldo Teixeirab,Paulo Borba. "Making Refactoring Safer through Impact Analysis." Science of Computer Programming. 2013.