ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 5, Issue 8, August 2016

Impact of Aspect-oriented Programming on Cross cutting Metrics using Breshman Technique for Homogeneity Quotient

Aarti Hans

ABSTRACT

Aspect-oriented is new programming approach to develop software. There are various existing approaches like modular and object oriented but these approaches suffer from limitation in properly separating crosscutting concerns. Examples of cross cutting concerns are caching, tracing, logging, security, resource pooling, synchronization, exception handling etc. This code must be properly encapsulated as well as must be scattered throughout the code. Through Aspect-oriented system tangling of code will be reduced which makes it easier to understand core functionality.We will use the Breshman line technique to reduce the complexity of code. However, the cohesion measure did not behave in a way that could be correlated to the actual changes performed on the code. A closer analysis showed that moving and merging of functionality could result in either an increase or a decrease in cohesion.

Keywords: AOSD, software estimation, crosscutting, breshman line.

I.INTRODUCTION

Aspect-oriented software development (AOSD) is a programming paradigm that addresses crosscutting concerns: features of a software system that are hard to isolate, and whose implementation is spread across many different modules. Well-known examples include logging, persistence, and error handling. Aspect-oriented programming captures such crosscutting behavior in a new modularization unit, the aspect, and offers code generation facilities to weave aspect code into the rest of the system at the appropriate places. Aspect mining is an upcoming research direction aimed at finding crosscutting concerns in existing, non-aspect oriented code. Once these concerns have been identified, they can be used for program understanding or refactoring purposes, for example by integrating aspect mining techniques into the software development tool suite.

1. Crosscutting metrics

Most of the crosscutting concerns manifest in early development artifacts, such as requirements descriptions [3] and architectural models due to their widely scoped influence in software decompositions. They can be observed in every kind of requirements and design representations, such as use cases and component models. Over the last years, aspect-oriented software development (AOSD) has emerged with the goal of supporting improved modularity and stability of crosscutting concerns throughout the software lifecycle. However, the use of aspect oriented decompositions cannot be straightforwardly applied without proper assessment mechanisms for early software development stages. This became more evident according to recent empirical studies of AOSD based on source-code analysis (e.g.) First, not all

types of crosscutting concerns were found to be harmful to design stability.

2. Bresenham's line

Even with today's achievements in graphics technology, the resolution of computer graphics systems will never reach that of the real world. A true real line can never be drawn on a laser printer or CRT screen. There is no method of accurately printing all of the points on the continuous line described by the equation y= mx + b. Similarly, circles, ellipses and other geometrical shapes cannot truly be implemented by their theoretical definitions because the graphics system itself is discrete, not real or continuous. For that reason, there has been a tremendous amount of research and development in the area of discrete or raster mathematics. Many algorithms have been developed which ‘‘map’’ real-world images into the discrete space of a raster device. Bresenham's line algorithm (and its derivatives) is one of the most commonly used algorithms today for describing a line on a raster device.

II.PROBLEM STATEMENT

The aim of the paper is to provide an experimental analysis over current testing methods for aspect-oriented software.

The current testing methods are useful in the sense that it points out strengths and weaknesses with different approaches by comparison. This information may help testers decide on how to test aspect-oriented software. Furthermore, an overview can point out new research areas as well as limitations of current research.

Research in testing of aspect-oriented software is still going on. Some researchers (e.g., Zhou et al., 2004) are proposing only a first step towards a comprehensive approach for testing aspect-oriented software. We also get some testing approaches for aspect-oriented program (e.g., Anbalagan&Xie, 2006) which only highlights some specific features. Aspect-oriented software introduced some new kinds of faults which we did not see in procedural program and object-oriented program. There is no coherent overview of the strengths and weaknesses of different test methods which can help testers and researchers to get idea about the current methods. There are some surveys, which discuss the effectiveness of some of the test methods. The effectiveness of the test methods is examined by their capability of fault finding. But until now, more test methods for aspects-oriented software are proposed.

III. SYSTEM MODEL

THE CONCEPTUAL COHENSION OF CLASSES

In order to define and compute the C3 metric, we introduce a graph-based system representation similar to those used to compute other cohesion metrics. We consider an OO system as a set of classes C = {c1; c2 ... *cn}*. The total number of classes in the system Cisn= |C|. A class has a set of methods. For each class c € C, M(c) = {m1.....; mk*}* is the set of methods of class c.

An OO system C is represented as a set of connected graphs GC = {G1;...;Gn} with Gi representing class ci. Each class *ci* € C is represented by a graph Gi € GC such that Gi = (Vi; Ei), where Vi = M(ci) is a set of vertices corresponding to the methods in class q, and Ei € Vi x Vi is a set of weighted edges that connect pairs of methods from the class.

DEFINITION 1

CONCEPTUALSIMILARITY BETWEEN METHODS (CSM).

For every class ci 2 C, all of the edges in Ei are weighted. For each edge (mk ;mj) € Ei, we define the weight of that edge CSM(mk ;mj) as the conceptual similarity between the methods mk and mj The conceptual similarity between two methods *mk*and *mj*, that is, CSM(mk;*mj)*, is computed as the cosine between the vectors corresponding to *mk*and *mj*in the semantic space constructed by the metrices method.

CSM (mk, mj) = vmktvmj / |vmk|2 |vmj|2 Where *vmk*and *vmj*are the vectors corresponding to the mk; mj 2 M (Ci) methods, T denotes the transpose, and vmkj 2 is the length of the vector. For each class c 2 C, we have a maximum of N = C2 distinct edges between different nodes, where n = |Mc)|. With this system representation, we define a set of measures that approximate the cohesion of a class in an OO software system by measuring the degree to which the methods in a class are related conceptually.

DEFINITION 2

(AVERAGE CONCEPTUAL SIMILARITY OF METHODS IN A CLASS (ACSM)

The ACSM c € C is ACSM(c) = 1/N x *ΣCSM (mi ,mj)* Where (mi; mj) € E, i ≠ j, mi , mj€ M(c), and N is the number of distinct edges in G, as defined in Definition 1.

DEFINITION 3 (C3)

For a class c € C, the conceptual cohesion of, C3(c) is defined as follows C3(c) = {ACSM(c), if ACSM(c)>0, Based on the above definitions, C3 (c) € [0, 1] V c € C. If a class c 2 C is cohesive, then C3(c) should be closer to one, meaning that all methods in the class are strongly related conceptually with each other (that is, the CSM for each pair of methods is close to one).

IV. PROPOSED IMPLEMENTATION

In our research, we will focus on the testing of Aspect oriented programs with crosscutting state based and Breshhman line. Particularly for testing we will use MATLAB tool. In our research we will focus on faultsfinding for Aspect Oriented Programs with help offlow diagram based on state base of crosscutting. Then testsequences will be generated based on the attributesavailable in input Aspect oriented code. Testsequences will also be based on interaction betweenaspects and primary models, and verifies theexecution of the selected sequences.

In our researchwe will propose Breshhman line and crosscutting metrics, which can detect incorrect advice type errors, weak pointcuts, and incorrect Precedence errors. Sequences will begenerated by proposed scheme automatically whiledetection of faults in Aspect Oriented codes whichwill validate the coming results.

The major steps of our method are described in thefollowing:

1. Building an aspect oriented code with theseparation of crosscutting concerns.

2. Building state based model of the primary concern

3. Testing the primary concern separately.

4. Integrating an aspect. As long as there are aspectswhich are not integrated
a. Building aspect model and weave it into primarymodel.
b. Generating the test sequences affected or createdby the aspect.
c. Testing the primary concern with the integratedaspect and executing it.
d. If there is no problem encountered, return to

Step 4.Testing entirely the primary concern includingaspects.
5. End.

4.1 Bresenham's line

We can now write an outline of the complete algorithm.

1. check input line up to endpoints, (X1,Y1) and (X2, Y2)
2. Calculate constants:
Dx = X2 – X1
Dy = Y2 – Y1
2Dy
2Dy – Dx
3. Assign value to the starting parameters:
k = 0
p0 = 2Dy – Dx

4. Plot the value at ((X1,Y1)

5. For each integer x-coordinate, xk, along the line if pk< 0 plot pixel at (xk + 1, yk)
pk+1 = pk + 2Dy (note that 2Dy is a pre-computed constant)

else plot pixel at (xk + 1, yk + 1)
pk+1 = pk + 2Dy – 2Dx
(note that 2Dy – 2Dx is a pre-computed constant)
increment k
while x k < X2

WORK FLOW

Start =>Aspect code => MATLAB code => use crosscutting matrices & Breshman line code=>Reduce visible fault => Testing outcome on command window => Test sequence generation => Finding faults=> Correcting faults=> Get graph result of crosscutting matrices.

V.RESULT

Aspect_Based_Features =

core: 0.2882

c1: 1.0457

c2: 0.9472

c1_h: 0.4404

c2_h: 0.3563

f: 0.1400

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\C1.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\C1_h.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\C2.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\C2_h.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\F.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\bresenham_line.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\codesize.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\main.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD -Arti_ITM_21.11.2015\sloc.m...

D:\M.tech_THESIS\MATLAB_CSE& ECE_PROJECT\AOSD_ITM_AARTI_HANS\ AOSD - Arti_ITM_21.11.2015\viewClassTree.m...

Result 1: Path of the program file

Total files: 10

Total lines: 553 (avg. 55 per file)

code: 396 (72%, avg. 40 per file)

comment: 65 (12%, avg. 7 per file)

QoC: 92 (17%, avg. 9 per file)

Total characters: 14318 (avg. 26 per line, 1432 per file)

code: 8565 (60%, avg. 22 per code line, approx.)

comment: 5856 (41%, avg. 90 per comment line, approx.)

Result 2: Total reviews of the code

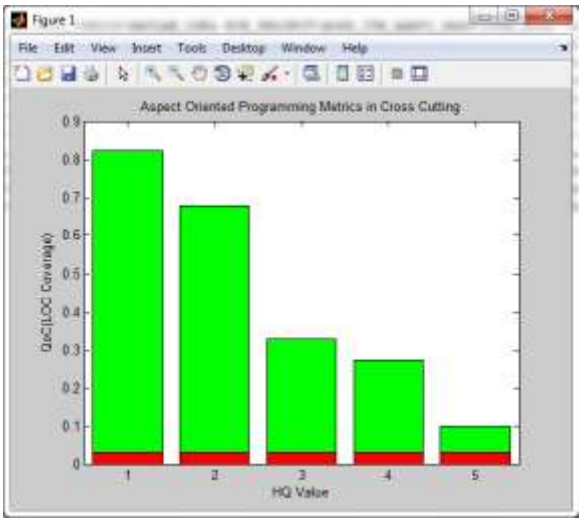

Figure 1: HQ value for line of code improvement.

In above figure a feature of a program P, we define the following metrics:

FCD (Feature Crosscutting Degree)

In Corresponds to the number of classes that are crosscut by all pieces of advice in a feature and those crosscut by the ITDs.

FCD(f,P)=count(union(ccclasses(method ITDs(aspects (f))),

Ccclasses (constructor ITDs (aspects (f))),

Ccclasses (field ITDs (aspects (f))),

sclasses (shadows (pointcuts (advices (aspects (f))),P))))

ACD (Advice Crosscutting Degree)

In Corresponds to the number of classes that are crosscut exclusively by the pieces of advice in a feature.

ACD(f,P)= count(sclasses(shadows(pointcuts(advices(aspects(f))),P))) HQ. We define Homogeneity Quotient as the division of the advice crosscutting degree (ACD) by the feature

crosscutting degree (FCD): HQ(f,P) = ACD(f,P)/FCD(f,P) if FCD(f,P)!=0 = 0 otherwise

PHQ(Program Homogeneity Quotient)

It corresponds to the summation of the homogeneity quotients for all the features in a program, divided by the number of features NOF. PHQ(P) = sum(foreach(P, λg.HQ(g,P)))/NOF(P)

VI.CONCLUSION

Assessing the quality of software is an essential process of software engineering. Theproblem of separation of concerns fueled the growth of the aspect-oriented paradigm.This new paradigm raises questions about quality, due to its close relations withobject-oriented programming. It is apparent from this paper that implementation of Designing an application in aspect Matlab framework is much sorted out in comparison to object-oriented concepts. As it separates out the functional code and non-functional code, so the complexity reduces of the software development.

We plan to extend our set of metrics to address issues such as cohesion and coupling for features. These extended metrics could help identify opportunities for feature refactoring. Such a study has been beyond the scope of our work to date, although we hope to do one in the future. In the meantime, we have developed one initial measure of the degree to which applying AOP techniques can simplify an application.

REFERANCES

[1] EsubalewAlemneh "Current States of Aspect Oriented Programming Metrics" International Journal of Science and Research (IJSR)ISSN (Online): 2319-7064 Volume 3 Issue 1, January 2014
www.ijsr.net

[2] Roberto E. Lopez-Herrejon "Towards Crosscutting Metrics for Aspect-Based Features" Computing Laboratory
Oxford University
Oxford, England, OX1 3QD
rlopez@comlab.ox.ac.uk

[3] Zhou, Yuewei, Ziv, Hadar& Richardson, Debra (2004), "Towards A Practical Approach to Test Aspect-Oriented Software". *Workshops on Testing* of Component-Based Systems (TECOS ... GI *2004* P-58, 1-16 (*2004*)

[4] Parizi, Reza Meimandi&Ghani, Abdul Azim (2007), "A Survey on Aspect-Oriented Testing Approaches", Fifth International Conference on Computational Science and Applications.

[5] Zhao, J. "Towards a Metrics Suite for Aspect-Oriented Software". Technical-Report SE-136-25, Information Processing Society of Japan (IPSJ), March 2002.

[6] Mehner K., "On using Metrics in the Evaluation of Aspect-Oriented Programs and Designs", 2005.

[7] Bartsch, M., Harrison, R, "An Evaluation of Coupling Measures for AspectJ", LATE Workshop AOSD (2006).

[8] C. Sant'Anna, Alessandro Garcia, Christina Chavez, Carlos Lucena& Arndt von Staa, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework". XXIII Brazilian Symposium on Software Engineering, Manaus, Brazil, October 2003.

[9] Jean-François Gélinas, MouradBadri, Linda Badri, "A Cohesion Measure for Aspects",

Journal Of Object Technology, Vol. 5, No. 7, pp 97-114, September- October 2006.

[10] Apel Sven, Batory Don, Rosenmuller Marko. "On the Structure of Crosscutting Concerns:Using Aspects or Collaborations?", 2007