ISSN (Online): 2455-7838
SJIF Impact Factor : 6.093

EPRA International Journal of

Research & Development

(IJRD)

Monthly Peer Reviewed & Indexed International Online Journal

Volume: 4, Issue:5, May 2019

Published By
EPRA Publishing

SJIF Impact Factor: 6.093 Volume: 4 | Issue: 5 | May| 2019 ISSN: 2455-7838(Online)
EPRA International Journal of Research and Development (IJRD)
Peer Reviewed Journal

TO IDENTIFY EARLY ASPECTS IN DESIGN PHASE USING GOAL DRIVEN CLUSTERING APPROACH IN ASPECT ORIENTED DEVELOPMENT

Aashima Anand
Maharishi Ved Vyas Engineering College, Kurukshetra University

Mohinder Singh
Maharishi Ved Vyas Engineering College, Kurukshetra University

ABSTRACT

Aspect-Oriented Programming (AOP) is a paradigm that offers a novel approach to improve the modularity of system by allowing the separation of crosscutting concerns (CCC). The concerns identified in earlier stages are known to be early aspects. Concerns in requirement level augment the concern in design phase and then to implementation phase and so on. The main motive of this proposed work is to identify the goals and early aspects from class & method relationship in architectural design phase. To get the efficient work, a goal driven approach is used to model the requirements with the goals and a clustering technique is used to cluster those goals in pair wise manner and find the similarity degree. The identified aspects are further refactored and transformed into AOP language.

INDEX TERMS— *Aspect oriented Programming (AOP), Cross cutting concerns (CCC), Goal-Driven Approach.*

I. INTRODUCTION

In software engineering, breaking down a bulky large Isoftware system into minor distinct parts is a necessary way of managing the complexity and evolution of systems. Such a decomposition results in a ―separation of concerns‖. AOP [1] better aim to advance the modularity of the software system by identifying the aspects in early stages. Aspect-oriented Software Development (AOSD) targets the implementation phase of software development life cycle: developers find and capture aspects primarily in source code. But aspects can be resolved much earlier in the life cycle, such as during requirements phase and architectural design phase [2][3]. In the early phases of the software development life cycle, the early aspects are concerns that cross the central modules of an artifact resultant from the principle of separation of concerns. Due to the increase in complexity and obvious change in requirements, the aim of paradigm is to enhance the development of software; it supports implicit quality attributes such as reusability, ease of change and understandability of software system. Here, ―Early‖ refers to happen before execution in the development phase. An aspect is known to be a concern that can crosscut requirements artifacts in requirements phase and architecture design artifacts in design phase. Identifying early aspects from the software life cycle helps to improve modularity in the requirement phase and design phase. In [4], a tool is created to identify CCC using clustering technique which measures the similarities between different requirements and also the hierarchical algorithm is used to cluster those requirements. Concerns denoted by high priority terms are identified during starting clustering phase, while others denoted by less priority terms identified in following phases. The generated clusters

are further measured using metrics which measure the cohesion between clusters, physical dispersion across requirements and also the interaction between them. Search Based Technique and Differential File Comparison Algorithm (DIFF) are combined to analyze and identify CCC in Agent Framework [5].

A concept of software hybrid re-engineering can also be applied to the system code at early stages to reduce cost and moderate risk of system maintenance [6].To identify aspects in refactoring process, aspect mining and identification has to be conducted and then afterwards aspect implementation is conducted. The approach describes the CCC in Agent Framework and how to clean the framework by removing concerns through refactoring process. To avoid scattering and tangling, design and coding must be well organized which can be achieved by Aspect-oriented (AO) approach [7]. In addition, detecting CCC's at one stage offers benefits downstream. Awareness of requirements-level concerns helps the developer plan a better system, and knowing architecture-level aspects automates more robust implementation. Early aspects can span development activities, and many identify their way into the code level as traditional implementation aspects.

By increasing the modularity, the impact on system changes when modified can be reduced [8]. More concretely, detecting early aspects through phases can:

(1) Increase the stability and reliability of requirements and architecture designs with implementation as well as with each other. (2) Provide a validation, logic and traceability for aspects. (3) Help confirm that CCC in a system's domain is encapsulated as aspects at the time of execution. In AOP, the main problem is crosscutting concern (CCC) which hinders the evolution and modularization of program. AO approach provides a key for CCC problems in Object-oriented approach. The design started from architectural level to get comprehensive models. To ensure clean modal, conversion process from design phase into implementation (code and maintenance) phase in development becomes an important part in the discussion. Early aspects are concerns that intersect the software's problem domain, with the possibility for a dynamic influence on queries of scoping, arranging, listing and design. Evaluating early aspects advances early stage decision making, and assist suggest subjective benefits in the whole software development life cycle. However, exploration of early aspects is difficult because subjective are often unclear about the concepts involved, and may use different terminologies to direct their concerns. The aspects are scattered and tangled across the system which hinder the evaluation of system. Scattering is when same code is spread all through many program modules. Its implementation is not modular hence affect the multiple modules [9]. Tangling is when more than one concern is implemented in same module, hence making it difficult to understand AspectU [10] is an aspect oriented language which augments the use case model by introducing pointcuts, joinpoints and advice. The interaction of the CC behavior to the principal use-case model specified by joinpoint model which is defined by following constructs in AspectU: (1) joinpoints are any points where event occurs in model, (2) set of joinpoints are defined by pointcut. (3) The behavior is affected by advices at the joinpoints. Concerns within a use case model are modularized by AspectU.

II. ASPECT ORIENTED CONCEPT

AOP is divided into two parts: (1) Base code- The language element that defines the basic functionalities of the system is stored in this code. It includes the core definition for the sequence of the implementation. (2) Aspect code- The aspect that encapsulates concern is stored in this code. It can add extra functionalities to the core base code and also control the flow [11]. The concept of AOP defines the mechanism of the concern handling. Join point is any point in program where an event occurs such as method execution, exception handling, field access etc. Pointcut is a collection of joinpoints, which defines where exactly an advice should be applied. At a particular join point, an action is taken before or after the implementation of joinpoint, which is represented by advice. Weaving is a process to link aspect with other app at compile time. Basically, this refers to the connection between the base code and aspect code. Fig. 1 depicts the example of AOP AspectJ Hello Program, it includes the joinpoint. Whenever an event occurs,in our example, calling greeting method, sayHello method and instantiating an object are described as joinpoint. This is described as the base code describing the core functionalities of the system.

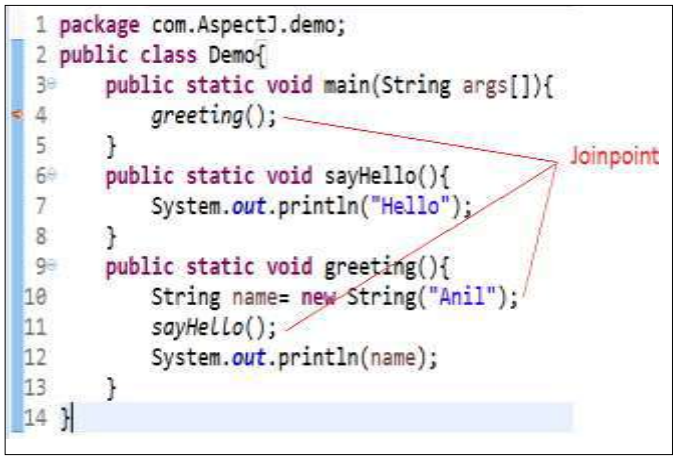

Fig.1 Aspect oriented Programming

In AOP, alter in the core code can lead to the scattering of the join-points across the system. [12] developed an approach to alleviate the burden caused by fragility. The automated technique is based on harnessing arbitrarily deep structural commonalities. The approach deals with the problems to rejuvenate broken pointcuts.

Fig.2 illustrates the same example of Hello Program. A pointcut callsayHello() is defined which calls the greeting method. An advice is also used before and after the trigger of pointcut. Advice before with Before Call message executes before and advice after with After Call message executes after the execution of pointcut. AORE

[13] detect candidate aspects by demonstrating the relationships between requirements and concerns in the form of matrix based on the scale of negative or positive impact of each aspect on others.

```
1 package com.AspectJ.demo;
2 public aspect Hello{
3     pointcut callsayHello():call(* Demo.greeting());
4     before():callsayHello(){
5         System.out.println("Before call");
6     }
7     after():callsayHello(){
8         System.out.println("After call");
9     }
10 }
```

Advice

Fig. 2 Advice in a program

Conflicts with stakeholders are solved by prioritizing concerns. The requirements specification is then revised based on the new priorities. Early-AIM adopts natural language. An automatic approach is defined to restructure the identified concerns into aspects. Approach used the Association Rule Mining techniques to automatically suggest the appropriate refactoring based on the identified aspect and Hidden Markov Model to additionally restructure the program to preserve the behavior of the system and also to lower the burden of developer [14].

III. SYSTEM MODELLING & DESIGN

To implement the proposed work, a simulation tool is designed in a popular NetBeans IDE. With the help of ArgUML tool, architecture of requirements is designed. The design is exported in xmi (xml metadata intermediate) format, which will be taken as input in our simulation tool. The tool reads all the interactions and relationship among the goals from the xmi file. After triggering the Discover Aspect button, the algorithm implemented on the provided input and the results will be displayed on the Log.

Fig.4 Simulation Tool

Transform to AOP transform the identified aspects into AOP language construct. Fig. 4 denotes the design of the simulation tool. This work also uses another tool ArgoUML. Interaction between Goals and Use Cases helps in providing vital details for managing, identifying and justifying software requirements. Use Cases are derived based on goal interaction. In our approach, goals are identified based on their interaction with use case. They are clustered using the clustering technique and then the similarity is checked across the clustered goals. Clustered goals with similarity degree above the defined parameter are further processed to identify the aspects. The identified aspects are transformed into aspect-oriented language construct. Fig. 5 illustrates the flow of proposed work in which goals are clustered together to find early aspects. If the clustered goals have similarity degree (common concerns), it is further processed for aspect identification otherwise it will be directed out of the process.

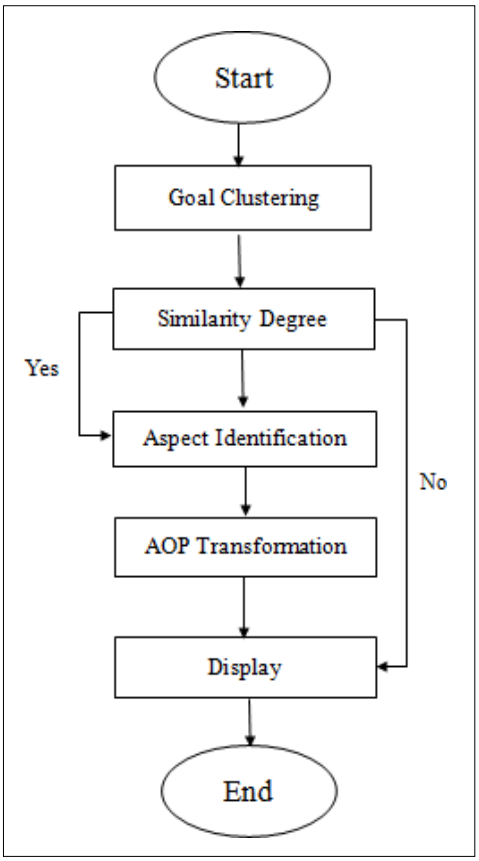

Fig. 5 Flow Chart of Aspect Identification

A. Goal Driven Approach

Goal Structure represents the goals to understand the various facets of functional and non-functional requirement as well [15]. In goal driven approach [16],

goals are used to represent the motive of the requirement. For ex- Friend plans his birthday party time and location by asking his parent. In a requirement statement, a verb in the sentence

B. Clustering Techniques

To overcome the amount of data items and to group similar data items, clustering is done [18]. Hierarchical clustering is used in the proposed approach, as it moves in stepwise manner. In this technique each goal is represented as a separate cluster itself and merges themselves in a pair. Because a cluster is itself combining in a pair and forming a big cluster, it is also following agglomerative method. Fig. 6 depicts the clustering of goals with common concerns captured. Similarity is measured between the clustered goals and based on the similarity degree, the aspects are identified. Similarity denotes the degree of correspondence among goals across all the characteristics used in the analysis.

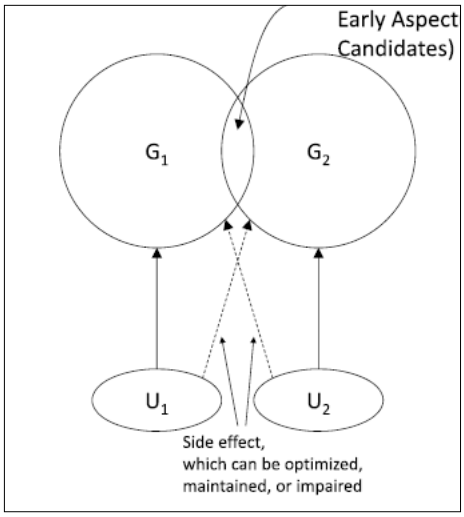

Fig. 6 Goals with crosscutting

C. AOP Implementation

AOP helps in achieving the modularity by improving the separation of concerns [19]. It consists of pointcut and aspects are encapsulated into a single unit. AOP language can also use advices which are triggered just before or after the joinpoint. It also includes types of advices which are executed based on the condition of pointcut. The identified aspects are transformed based on static name based pointcut. As it is the initial stage of the life cycle, the static pointcut is valid for the aspects.

can be used to depict the goal. Here, party represents the goal. In goal driven use case (GDUC) approach, the use cases are viewed as process to achieve particular goals. In GDUC Model [17], goal identification and formulation is done. After the identification, GDUC diagram is designed in which each use case is observed as a procedure that can be linked with the goal to be achieved.

D. Proposed Algorithm

The algorithm is designed to identify the goals from the design phase by the mean of xmi file and perform hierarchical clustering technique to identify aspects.

Input: Goal file with concerned use case.

Output: Early Aspect Discovery.

Initialize:

I. Identify the goals from the requirement to be clustered.
II. Cluster the goals in a pairwise manner (Gi, Gj). Gi is not equals to Gj.
III. Identify the similarity between the goal Gi and goal Gj.
IV. If the similarity degree between Gi and Gj is zero or null, return to Step VII.
V. If the similarity degree between Gi and Gj is not equal to zero, set a co-efficient value (m) to filter out the required goals.
VI. Identify the early aspects (An).
VII. End.

E. Requirement & Goals

Consider a scenario of Drawing Editor with the following Requirements:

- Artist can start, open document editor to create sheet and document.
- Artist can create, open sheet.
- Artist can create, open document.
- Document consists of groups of drawing object and geometric object.
- Drawing object includes text figure.
- Geometric object includes circle figure.
- Geometric object includes rectangle figure.

Based on these requirements, the goals are set. Requirements are the process to achieve the goals. For joinpoints which is triggered in the case of concerns occurred. The aspect identified in design phase is transformed into AOP language construct. The identified\

Ex. Artist can create a sheet. Here, sheet is goal which can be achieved by a process (requirement) create. Fig. 7 depicts the goal-driven use case modelling diagram of

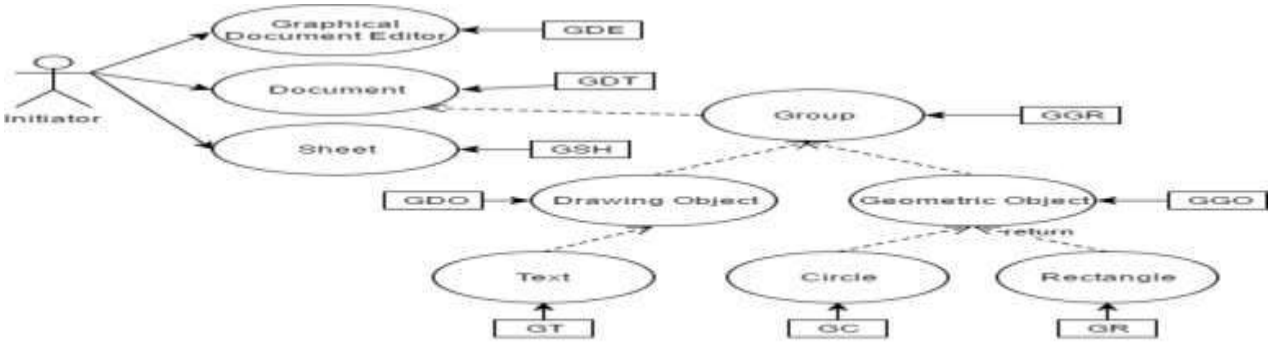

Fig. 7 Use case representing requirements and goals

Acronym	Goal Name
GGD	Graphical Document Editor
GDT	Document
GSH	Sheet
GGR	Group
GDO	Drawing Object
GGO	Geometric Object
GTX	Text Figure
GCR	Circle
GRT	Rectangle

Table 1: Acronyms for goals name

All the details are taken from the GDUC and the above presented acronym table, a class-method relation diagram is designed in ArgoUML tool to represent the goal-requirement interactions. Based on the relation diagram, a goal requirement interaction class diagram is designed as shown in Fig. 8.

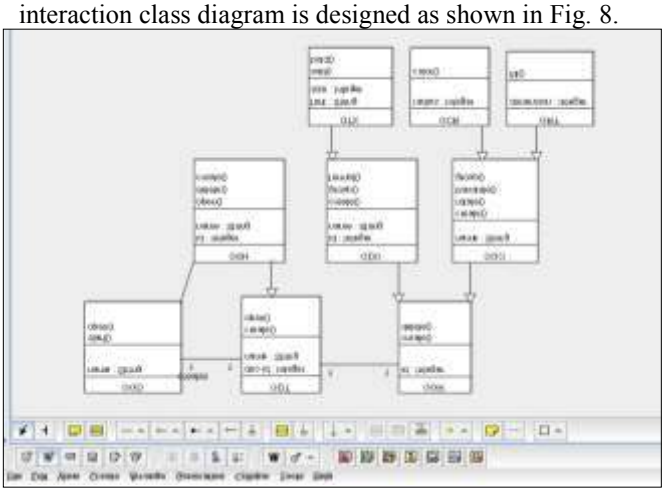

Fig.8 Class Diagram representing Goal Driven Approach.

The above diagram depicts the class diagram of the drawing system in which each goal is interacted with the other goal. Further the information of the design phase is wrapped up in xmi file. The simulation tool will identify the goals and then execute hierarchical based clustering on their interaction distance to find the similarity and also to identify the aspects.

IV. RESULTS & DISCUSSION

In drawing system, the main objective is to find the early aspect. A goal driven approach is used to identify the aspects [20]. In the presented simulation tool, the goals with similarity degree are identified and highlighted as red. The parametric value (m) is taken different values based on high-priority and low-priority goals and concerns identification. Fig.9 illustrates the concerns that are identified from the goals by assuming high priority aspect identification. Aspects identified as high-priority may highly affect the modularity of the system. Shown in results, a high value of parameter is taken i.e. 0.9. The value ranges from 0.0 to 1.0. As shown in figure, each goal clusters are clustered with other goals in a pair. Similarity degree is measured among the clustered goal. The clusters having degree greater or equal to set parameter is highlighted and also the aspects that crosscut their functionality is identified and displayed in the results. For example- The concerns create and open are similar in goals GDT & GSH. Also, the concerns create and delete are similar in goals GGR & GSH which hinders the modularity of the system. These concerns should be further refactored, so as to improve separation of concern and modularity.

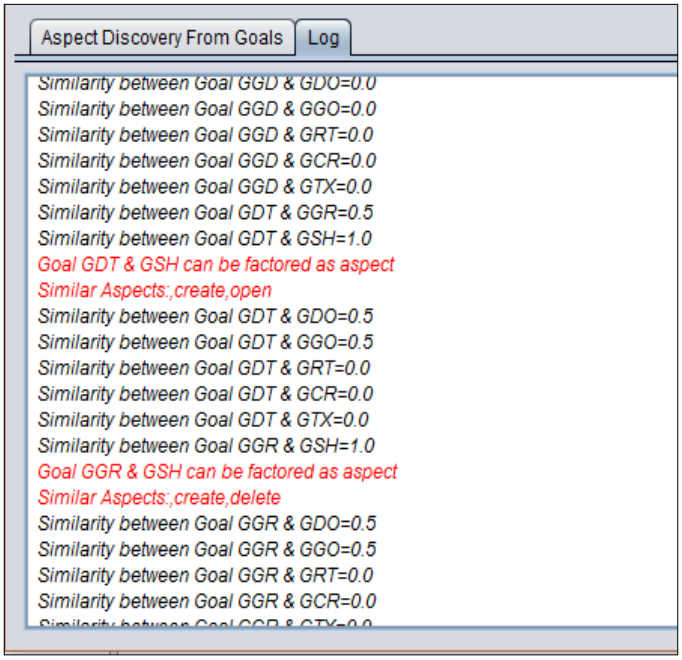

Fig. 9 Identified aspects

A. Refactoring & AOP Transformation

The identified aspects are restructured and transformed into AOP language. By refactoring, the modularity can be improved. Whenever the concerns that crosscut the functionality of other goals are identified, a special method or function must be executed to prevent the system tangling. A language named AspectJ, which is based on AOP, is used. The concept of AOP (AspectJ) is discussed in section 2. Fig 10 illustrates the AOP language construct, which has a pointcut named ptName. The pointcut triggered the call method. Restructuring is done to make pointcut more dynamic, the expression G* represents the dynamic goal with the identified aspects. Advices before and after are execute with the pointcut. The output of the AOP construct is extracted in a text file.

```
public aspect Goals{
pointcut ptName():call(* G*.create,delete);
before(): ptName(){
System.out.println("Execute before call method");
}
after(): ptName(){
System.out.println("Execute after call method");
}
```

Fig.10 AOP Construct

According to the parameters taken based on the priority, the coefficient value of m is taken different values to get the results. At different value of m, the goals with priorities are clustered to identify the aspects. The results are taken by setting the value of (1) m > 0.9 and (2) m=0.5

Goal 1	Goal 2	Aspect Identified (An)	No. of Aspects
Setting coefficient value (m) > 0.9			
GDT	GSH	create, open	2
GGR	GSH	create, delete	2
Setting coefficient value (m) = 0.5			
GGD	GDT	open	1
GGD	GSH	open	1
GDT	GGR	create	1
GDT	GDO	create	1
GGR	GGO	create	1

Table 2 Results showing identified concern

Table 2 results in the goals interaction and identified aspects from the design phase. Each goal is clustered with other goal. The aspects are identified from the goal cluster. For example- In goal clustering of GGR and GSH, the identified aspects are create and delete and also the number of aspects identified is represented in the table. Previously in [21] the aspects are identified in requirement phase by clustering the goals at large scale. Our approach augments the work by identifying the aspects in architectural design phase and also to transform those aspects into AOP language construct to avoid CCC and to advance the principal of separation of concerns.

V.CONCLUSION & FUTURE WORK

Aspect-oriented programming provides a mechanism to avoid code tangling and scattering problems caused by crosscutting concerns and also to modularize the concerns. Concerns found in early stages can also affects the further stages of software development life cycle. To effectively support the work it is imperious to offer an approach which will reduce the burden of the developer by identifying and structuring base concerns at early stages which provides benefit at the later stages. This paper proposed an approach for identifying the concerns in the early stages of software lifecycle such as architectural design phase and translating them into AOP language. The identified aspects are extracted from the design phase of goals and requirement relationship. Our approach addresses two issues: (1) Identifying the early aspects, (2) restructure them into AOP construct. This work concluded that the aspects can identified in architectural design phase and also can be weaved separating those concerns in AOP construct using AspectJ.

There are many interesting challenges left for future work. Our future work include: (1) Automatic refactoring process of identified concerns. (2) Additional restructuring of the software by removing bad smell in identified concerns.

REFERENCES

[1] *G. Kiczales, Gregor, and Erik Hilsdale. "Aspect-oriented programming." ACM SIGSOFT Software Engineering Notes. Vol. 26. No. 5. ACM, 2001.*

[2] *J. Mylopoulos, L. Chung, and B. Nixon, –Representing and using nonfunctional requirements: A process-oriented approach,‖ IEEETrans. Softw. Eng., vol. 18, no. 6, pp. 483–497, Jun. 1992.*

[3] *A. van Lamsweerde, R. Darimont, and E. Leitier, –Managing conflicts in goal-driven requirements engineering,‖ IEEE Trans. Softw. Eng., vol. 24, no. 11, pp. 908–926, Nov. 1998.*

[4] *C. Duan and J. Cleland-Huang, —A clustering technique for early detection of dominant and recessive cross- cutting concerns,‖ in Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design 2007., May 2007*

[5] *Nugroho, Lukito Edi, Widyawan Widyawan, and Ahmad Ashari. "Crosscutting Concerns Refactoring In Agent Framework." The 2nd International Conference on Information Technology, Computer and Electrical Engineering (ICITACEE 2015). 2015.*

[6] *Tarar, Sandhya, and Ela Kumar. "Design Paradigm and Risk Assessment of Hybrid Re-engineering with an approach for development of Re-engineering Metrics." International Journal of Software Engineering & Applications 3.1 (2012): 27.*

[7] *Sullivan, Kevin, et al. "Information hiding interfaces for aspect-oriented design." ACM SIGSOFT Software Engineering Notes. Vol. 30. No. 5. ACM, 2005.*

[8] *Garcia, Alessandro, et al. "Modularizing design patterns with aspects: a quantitative study." Transactions on Aspect-Oriented Software Development I. Springer Berlin Heidelberg, 2006. 36-74.*

[9] *Kellens, Andy, Kim Mens, and Paolo Tonella. "A survey of automated code-level aspect mining techniques." Transactions on aspect-oriented software development IV. Springer Berlin Heidelberg, 2007. 143-162.*

[10] *J. Sillito, C. Dutchyn, A. D. Eisenberg, and K. D. Volder, –Use case level pointcuts,‖ in Proceedings of European Conference on Object-Oriented Programming, 2004.*

[11] *Mens, Tom, and Tom Tourwé. "A survey of software refactoring." IEEE Transactions on software engineering 30.2 (2004): 126-139.*

[12] *Kellens, Andy, Kim Mens, and Paolo Tonella. "A survey of automated code-level aspect mining techniques." Transactions on aspect-oriented software development IV. Springer Berlin Heidelberg, 2007. 143-162.*

[13] *J. Sillito, C. Dutchyn, A. D. Eisenberg, and K. D. Volder, –Use case level pointcuts,‖ in Proceedings of European Conference on Object-Oriented Programming, 2004.*

[14] *Vidal, Santiago A., and Claudia A. Marcos. "Toward automated refactoring of crosscutting concerns into aspects." Journal of Systems and Software 86.6 (2013): 1482-1497.*

[15] *C. Rolland, C. Souveyet, and C. Achour, –Guiding goal modeling using scenarios,‖ IEEE Transactions on Software Engineering, vol. 24, no. 12, pp. 1055–1071, December 1998.*

[16] *Jonathan Lee, Kuo-Hsun Hsu, "GEA: A Goal-Driven Approach to Discovering Early Aspects", Software Engineering IEEE Transactions on, vol. 40, pp. 584-602, 2014, ISSN 0098-5589.*

[17] *J. Lee and N. Xue, —Analyzing user requirements by use cases: A goaldriven approach,‖ IEEE Software, vol. 16, no. 4, pp. 92–101, July/August 1999.*

[18] *Duan, Chuan, and Jane Cleland-Huang. "A clustering technique for early detection of dominant and recessive cross-cutting concerns." Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design. IEEE Computer Society, 2007.*

[19] *Coady, Yvonne, et al. "Using AspectC to improve the modularity of path-specific customization in operating system code." ACM SIGSOFT Software Engineering Notes. Vol. 26. No. 5. ACM, 2001.*

[20] *E. Baniassad, P. C. Clements, J. Ara_ujo, A. Moreira, A. Rashid, and B. Tekinerdo_gan, –Discovering early aspects,‖ IEEE Softw., vol. 23, no. 1, pp. 61–70, Jan.-Feb. 2006*

[21] *A. Sampaio, A. Rashid, and P. Rayson, –Early-aim: An approachfor identifying aspects in requirements,‖ in Proc. 13th IEEE Int.Conf. Requirements Eng., 2005, pp. 487– 488.*