Assessment of the Impact of Aspect Oriented Programming on Refactoring Procedural Software

Zeba Khanam,
S.A.M Rizvi,
Department of Computer Science,
Jamia Millia Islamia, New Delhi.
INDIA

Abstract---Many organizations suffer from superfluous, disproportionate, difficult-to-extend software because of the lack in maintenance effort and ignoring the fact that continuous refactoring provides a competitive advantage. Refactoring is assumed to positively affect the software parameters like scalability, modularity, reusability, complexity, maintainability, performance and efficiency. Several refactoring have been proposed for object-oriented languages, but there are few related works focusing on procedural programming. In this paper, an assessment is provided of selected literatures which relate to refactoring of procedural languages, and it also contributes to highlighting new concepts and requirements for developing new refactoring techniques for C that may eventually benefit other procedural languages also .To this end, we have studied the refactorings performed on 3 procedural languages that are Fortran, Cobol and C to analyze the pattern of refactorings followed in these languages and assess their cumulative effect in different applications. We offer a few refactoring capabilities that may improve the existing refactorings for these languages by our contribution to refactoring that characterizes of a new aspect oriented programming style.

Keywords: Reengineering, Refactoring, C, Cobol, Fortran, Procedural languages, Aspect oriented Programming.

I. INTRODUCTION

Legacy software is hard to maintain and understand due to various reasons such as those who have to maintain the systems are not the same who created them, the corporate strategy gets redefined, e.g. traditional data processing models are obsolete whereas multichannel, service oriented model is the preferred choice. Other reasons are business processes are redefined by management and business structure is reorganized, they have been built sometimes without anticipating that they would be still running decades later and so they don’t have the ability to change as rapidly without complicating the code.

Although refactoring concept has been born in the heart of the object oriented programming, it has crossed over these borders. Refactoring tools can be found in object oriented programming, in structured programming and functional programming. We begin by highlighting the contributions that have been made in the area of refactoring these procedural languages and that bear a relation to our work. In this paper we have analyzed the refactorings affect on the procedural languages that includes Fortran, C and Cobol. Thereafter, we conclude what are the major parameters that have been focused in those works and then we conclude that some parameters like modularity, scalability, reusability and performance of a code structure can be enhanced through refactoring by use of aspect orientation as we are working on refactoring C with AspectC. We begin by highlighting the work that has been done in the area of refactoring Fortran. The next section focuses on Cobol and then C refactorings.

.

II. ROLE OF REFACTORING IN FORTRAN

As a programming language, Fortran is one of the most ancient, yet it is still being used. Fortran is a procedural language heavily used in high performance computing. Its evolution has resulted in a wide range of equivalent syntactical constructions. From those equivalent constructions, the older ones (coming from old language version/s) have many disadvantages/drawbacks. The refactorings in Fortran are based on the traditional approaches.

The concept of refactoring as an interactive process performed by an expert programmer while carefully examining the code, in small and safe steps, was defined in Opdyke’s thesis many years ago, in this work refactoring is presented in the context of Object Oriented Programming. Thereafter the major contribution was by Ralph Johnson’s research group that promoted refactoring and the development of automated refactoring tools. The major development included introduction to Photran, an integrated development environment that was used to implement those refactorings , an automated refactoring tool for Fortran and high performance computing further discussed the impact of such a tool on legacy code reengineering [2][3]. Vaishali De has also identified 90 possible Fortran refactorings [5]. Later on, Overbey etal. [3] bring to light the need of refactoring tools integrated with IDEs for Fortran programs and in the High Performance world. John Brant and Don Roberts' Smalltalk Refactoring Browser [4] was the first implementation, introducing automated refactorings to the Smalltalk community.

A. *Major Refactoring approaches in Fortran Refactoring*

1) *Maintainability refactorings*

Most of the work in refactoring covers the parameters like scalability, modularity and maintainability. The paradigm viewpoint classifies the refactorings as Object Oriented refactoring, Structured Programming refactoring or Functional Programming Refactoring. But with Fortran, though it started as a procedural language, the later releases such as Fortran 2003 has made use of structured and object oriented features. Another viewpoint to adapt in order to create a good classification may be found in the way that users or programmers need refactorings. As a successful programming language Fortran is characterized as the oldest programming language for scientific purposes and having a huge production of legacy code due to its particular evolutionary process.

The major work in Fortran refactorings have specifically focused on 2 categories: *Refactorings to Improve Maintainability* and *Refactorings to Improve Performance*. Each one of these classes may be divided into subclasses [2]. However, the refactorings to improve performance is an entirely different class of refactorings that are unique to the domain of supercomputing. It is well-known that, despite the best efforts of compiler vendors, code intended to run on a specific supercomputer must undergo many hand optimizations. Examples include manual unrolling of loops and optimizing data structures based on the machine's cache size.

In the first place we refer to the most recent work done by Mendez in the area of refactoring Fortran. Fortran being a fifty-year-old programming language with a large number of software applications developed through years and with most of the Fortran software being legacy. The findings in [2] dealt with refactoring as a technique to understand, to comprehend, to upgrade, to modify and to add changes on legacy software. Some of the refactorings focused on parallelizing and performance improvements. Moreover the contribution to Photron project was also made. The parameters for refactoring were Improving Maintainability and Performance.

2) *Performance Refactorings*

In general the refactoring techniques focus on improving the external attributes like maintainability and understandability but performance based refactorings are generally not attempted. This category currently has two examples of how refactoring can be used to improve performance while preserving not only the behavior of the program but also the readability and maintainability of the code. Some major refactorings for the purpose in Fortran are Change To Vector Form, Interchange Loops, Loop Reversal, Loop Unrolling.

B. *Scenario in Fortran Refactoring*

There are certain refactoring that are common to most of the languages for example renaming methods, variables, extracting fragments of code etc. But many of the refactorings done in Fortran are very much Fortran specific. The reason is that Fortran has had a particular evolutionary process through different versions across time, about ten language versions have been published in the last 50 years (six of them were standards). These versions have transformed Fortran into a language with a rich set of syntactical constructions. Therefore each version corresponds to different set of refactorings. As a consequence, programs written years ago are hard to read because of the lack of modern software engineering concepts such as software quality, development processes, etc. As a result most of the work is focused on improving the code readability and understandability [16]. Therefore, as has been highlighted above the refactorings are focusing on avoiding poor Fortran practices, removing outdated and obsolete and non standard constructs that fall under the category of improving maintenance. The performance refactoring is generally dealt with interchanging and reversing loops. But what has not been focused maintainability refactorings are the internal software attributes like modularity and scalability. The refactorings that should primarily focus on increasing the cohesiveness and decreasing the module coupling is not explored broadly. There are certain concerns that crosscut the whole source code and increase the code entanglement thereby increasing the code duplication. This issue most conveniently can be handled by aspect oriented approach, but till now the aspect oriented version of Fortran has not evolved therefore this issue can be handled with the various versions of *Extract fragment to method* refactoring. In the area of automated support to refactoring a set of automated refactorings for Fortran based on the Photran plug-in are described in [21] that are meant to improve the design of existing applications.

Fortran has undergone a complex evolutionary phase. The upgradation of Fortran language to Fortran 2003 was a major revision that incorporated the object oriented features also such as: type extension and inheritance, polymorphisms, dynamic type allocation, and type-bound procedures. Therefore, some of the existing object oriented refactorings can also be used in the similar way as are applied to the existing object oriented software. Furthermore, the object oriented code smells may be used to identify the symptoms for refactoring.

III. Role of Refactoring in Cobol

Cobol is one of the oldest programming languages. Cobol was created in 1959 as a language that has its primary domain in business, finance and administrative systems for organizations and government. This focus led Cobol to become the preferred language for business development, starting in the 1960s. When implemented in software, business knowledge, information and rules tend to be spread out over the entire system. With applications written in Cobol this is even more the case, as Cobol is a language targeted at business processing but without modern day modularity mechanisms. This information tends to get lost over time, so that when some maintenance is required one is again forced into reverse engineering [31]. As the language evolved, standards emerged, and Cobol penetration in the business market increased. As Cobol is the traditional powerhouse for business applications, Cobol should expect significant growth. Experts instead have forecasted a decrease in Cobol development for the future. (This constant forecast has been published for the last 20 years, first stating that Cobol was to be overtaken by C, then by Java, etc.) Since 1959 Cobol is continuously in the evolving process. Cobol-68, Cobol-74, Cobol-85, Cobol- 2002. The Cobol-2002 included object orientation as its main feature. Though the language is in a continuous evolving process, still changes in technology and resources have lessened the need for Cobol specific processing during the last 20 years due to the lack of complex graphical screen designing, due to the availability of good DBMS its impact as a data processing language has reduced and also due to its lack of integrating capabilities with other non Cobol business applications.

Cobol is not a distributed and object-oriented language, however its integration with other languages or distributed systems is a prerequisite for achieving migration towards Web technologies. Therefore to make it compatible with the current technologies most of the research work focuses on migrating the legacy Cobol to web based architecture [10]. Legacy COBOL applications are now becoming a risk to your business. Most researchers now feel that moving the COBOL applications to another platform only prolongs this situation, since the application will still have the same functionality and problems. For example, you would still need to pay third-party vendors for compilers and runtime environments. Appending Web services and Web clients to your existing systems only increases the complexity of your existing architecture. Therefore most of the work is done to migrate away from the legacy COBOL to Java or COBOL to C#.

A. *Scenario in Cobol Refactoring*

Due to its structure Cobol has not been a much focused area for refactoring. The earlier versions of Cobol did not support local variables, recursion, dynamic memory allocation, or structured programming constructs. Support for some or all of these features have been added in later editions of the COBOL standard. COBOL is still the dominant language on mainframe computers. Most code is written once and read many times—usually, to focus on a particular point (for instance, to fix a bug). Thus, it is important that the reader quickly grasp the essence of what's happening. COBOL though is readable but, because of its verbose characteristics, many lines of code have to be read to get anywhere. Old-fashioned COBOL (in contrast to OO COBOL) tended to use PERFORMs, instead of procedure calls, and that means that the logic and the data are miles apart. C does not have this problem; but C can still be hard to understand. This is partly the fault of the language. But refactoring has not been focused in legacy Cobol code. The main reason for this could be that due to its unstructured nature Cobol refactoring is not easy and safe as JAVA/.NET refactorings. Though piecemeal refactorings are attempted at various levels in Cobol programming but a complete refactoring catalog is not available till now.

With the evolution of aspect oriented programming in Cobol a scope of refactoring has increased with the usage of aspect oriented programming constructs [11]. The presence of crosscutting concerns is an indication to code smells that can be refactored. A list of cross cutting concerns has been identified [11]; some of them are logging, tracing, context-sensitive error handling, coordination of threads, remote access strategies, execution metrics, performance optimization, persistence, authentication, access control, data encryption, transaction management etc. There are many cross cutting concerns that are definitely meaningful from the Cobol perspective. For example Logging certain file operations or subprogram and procedure executions is implemented in Cobol code on a regular basis — with varying degrees of tangling. Error handling is another cross cutting concern existing in the Cobol code. Another issue like synchronization is (much) does not effect the Cobol code much as the Cobol standard does not define expressiveness for multi-threading. Only the newer versions support it that is not widely used in business.

Another work includes application of aspect orientation in understanding Cobol code by defining the context aspect and error handling aspect [12]. Aspect orientation is also applied to strengthen the internal control in enterprise information systems. In this regard Altair environment was used for understanding and developing programs for aspect-oriented Cobol [23].The researchers claim that aspect orientation can be applied to strengthen the internal control of existing Cobol programs. Cobol companies worldwide are also making their information systems comply with the new standards and requirements such as the SOX like laws [23] [22].The additional features that needs to be implemented for this compliance are when implemented using aspects results in better modularization and separation of concerns. 2The business rules staying as it is in the base code and the additional features being implemented using thee aspects.

IV. Role of Refactoring in C

Though plethora of research work has been done on refactoring object-oriented languages. But there is still lack of refactoring tools when it comes to the C programming language. The reason is the presence of C's preprocessors directives to both parsing and ensuring the correctness of applied refactorings. But C still remains a dominant language with its widespread applications. Nevertheless, there are good reasons for exploring refactoring needs and thereby, new refactoring tool for C as it remains one of the most dominant languages in use and is widely used in a large number of legacy systems.

This section highlights the major research work done in the area of refactoring C. The first attempt in refactoring the software is by W. Opdyke and Ralph Johnson [60]. He proposed and described each of the primitive refactorings and proved the *preconditions* that must be met to ensure that the transformation preserves program behavior. The other major development was made by Fowler [Fowler 99] who had published a book on Refactoring. The book presents a catalog of refactorings, with examples in Java. However, he did not construct a tool implementing his ideas. Specifically in the area of refactoring C language code, Garrido seems to have introduced the refactoring concept to structured programming [6]. Her work is based on refactoring C programs [6] [7]. However, the major refactorings dealt with adding, deleting, changing a program entity and few other refactorings. The work also contributed to the development of a refactoring browser, CRefactory, with the objective of reusing the architecture of the Smalltalk Refactoring Browser to construct a refactoring tool for C. Her PhD thesis presented an algorithm to handle C preprocessor directives. An extension of the refactoring catalog has been attempted in [18] along with the survey of the tools existing for refactoring C code and their underlying design and implementation techniques.

Presently, one of the commercially used refactoring tool for C is XRefactory created by XRef-Tech [19]. It is a C/C++ refactoring browser which works with the Emacs and XEmacs editors and offers the following refactorings for C | extract function, rename program element, and delete or move parameters.

Another important tool in the world of C programming is the development of a syntactic replacement language for C ASTEC [17] with a translator tool Macroscope, which translates the directives into the new language that deals with the most important deficiencies of the preprocessor and an ASTEC aware refactoring tool that handles preprocessor constructs naturally. The refactoring browser CScout developed by Spinells[9], running on a powerful workstation, can be used to accurately analyze, browse, and refactor large program families written in C. CScout is designed to handle multiple related projects , collection of C source files that are linked together, also handling most of the complexity introduced by the C preprocessor. CScout takes advantage of modern hardware (fast processors, large address spaces, and big memory capacities) to analyze C source code beyond the level of detail and accuracy provided by current ideas, compilers, and linkers.

The most recent addition to the area of refactoring is the use of Aspect oriented programming in formulating new refactoring techniques. The use of AOP constructs can help solve the problem of crosscutting concerns as they are hard to implement and change consistently because multiple, possibly unrelated, locations in the code have to be found and updated simultaneously[20]. There is a prospect of widespread adoption of aspect orientation in C as that has already started in object oriented paradigms with the predominance of AspectJ. In this context, we have introduced new refactoring techniques with preconditions and steps using the aspect oriented constructs such as extract conditionals to advice, encapsulate fields, merge duplicate conditionals [13].

A. Enhancement Refactorings in C using AOP

It is necessary to reevaluate existing procedural refactorings because the constructs of AOP programming languages significantly affect what changes can be meaning-preserving. We are working on C refactorings that will be valuable to the programmers and is also different from the traditional refactorings as they incorporate the aspect oriented constructs also. We extracted from existing catalogs [8][14][15] the ones that can apply to C and added some novel refactorings too. In this section we give examples of a few refactorings using a case study that are novel to the C programming using the Aspect oriented constructs.

One of the major benefits obtained by Aspect oriented code is the extraction of cross cutting concern. In this section we explain the so-called crosscutting concerns that scattered across the various functions or over the whole code. Therefore apart from the AOP based refactorings for C, the other facet to refactoring the code is that it should be able to do enhancements or modifications, such as: This would increase the scalability of the system as well because one of the major aims of refactoring is to prepare the source code for future refactorings and extensions such as:

- Adding New Concerns
- Updating existing concerns
- Removing existing concerns.

In general, if the following operations are to be performed on a code it is a cumbersome and time consuming job but the AOP constructs simplifies the job to a great extent. The next section highlights how the following changes can be accommodated easily using the AOP constructs. For this purpose we have highlighted the scenarios taken from a case study "Mobile store Information System" that maintains the records of the its employees and customers.

1) *Addition of a new concern*

If with the current customer a change has to be accommodated that the customer details should also include the user ids so that they may be able to check their records online and pay their bills online. In this case the customer structure can be updated without disturbing the original structure and additional fields can be added using the introduce advice.

But why to use the introduce () advice? The utility of adding it using an introduce () advice is depicted if the task performed by the added parameter could also be captured by an advice that would not disturb the original source code and thereby the functionality provided by the new parameter may not be added in the original code but in the advice. This is shown in the code snippet below (Listing 1 and Listing 2)

```
Before Refactoring

struct customer
{
            char id[NUM];           /* ID of Length 5*/
            char name[MIN];         /* Name of Length 30*/
            char address[MAX];  /* Address of Length 61*/
            char phone[PH];         /* Phone Number of Length 12*/
            int  connection;        /* Connection Type of length 2*/
            int  day;               /* It is used to display the day of
length 2*/
            int  month;             /* It is used to display the month of
length 2*/
            int  year;              /* it is used to display the year*/
}rec;
```

Listing 1: The customer structure that needs updation

```
After Refactoring

Refactoring for addition of a new member to a structure


introduce(): intype(struct customer) {          <-- advice : add a member
char *  userid;}


before (struct customer* ptr): call($ add( struct customer *...)) &&
args(*ptr)
 {
printf("Enter the user id");    //addition of functionality related to userid

gets(ptr->userid);
---
}
```

Listing 2: Adding new parameter to a structure

2) Updation of existing concern

To the following case study if it is required that before viewing, adding or deleting any employee record the authentication of the user has to be verified then the system needs to be updated at various places. The typical usage of aspect based refactoring is in places where a cross cutting concern is detected, as it can be separated as a concern in an aspect. The updation can easily be performed by adding an advice.

```
before():call (void add()) || call(void view()) || (call (void delete()))
{
//Add code to verify password
verify();     //Or a function may be introduced for the verification
}
```

Listing 3: Addition of verification code in a before() advice

The code for verifying the password can be written in the advice or as has been shown in the above listing a separate function is called for the verification purpose.

3) Removal of Existing Concern

There are certain concerns that may not be required by the application though they were earlier built into the application for a purpose. For example, everytime the record of the employee is accessed for viewing , updation or deleting the status of the employee is prompted through a function named status ().Suppose the status checking concern is no more required, Instead a security concern is to be introduced . This requires modification at various places in the program disturbing the structures of various functions, but the modifications can be easily obtained by AOP constructs. As shown in the listing 4 below, the call to the status function may be avoided without removing it actually and doing modifications at various places by using the around advice that skips the calling of the function status () (Listing 5).

```
void view(struct employee c)       //Scenario displaying the calling
of                                 function status() after the call to
each function
{
status(c);
-
-
}

void update(struct employee c)
{
status(c);
-
-}

void delete(struct employee c)
{
status(c );
-
-
}
```

Listing 4: The original structure of the functions

```
void around(): call(void status(struct customer c))
{
printf("Status not displayed");          //call to status is omitted

securitychecking (struct customer c)   //May or may not be introduced

}
```

Listing 5: Addition of around () advice for status ( ) function

V. Discussion

New technologies like aspect orientation can enable legacy systems to be managed with modern techniques, or reused cost-effectively to deploy new systems. Our discussion in this section describes concepts in AOP from the world of procedural languages (C, Cobol, Fortran).The result is extending system lifetimes years into the future, thereby deferring expensive replacement costs, or reducing the costs of deploying new systems by enabling the recovery and reuse of long-proven business rules and data models, potentially saving up to millions of dollars.

One of the areas where the traditional refactorings have not focused much is the existence of crosscutting concerns in the code. Aspect-Oriented Programming (AOP) provides new modularization of software systems by encapsulating crosscutting concerns. Based on the analysis of the refactorings that have been done for Fortran. It can be deduced that most of the refactoring intends to improve internal quality attributes of the code such as: readability, understandability and extensibility (attributes that refactoring has been recognized to improve) and removal of obsolete features. Similarly in the area of C programming also the refactorings proposed emphasize more on readability and understandability and also development of C refactoring tool to carry on the refactorings in largely automated way. Though a number of researchers have focused on bringing aspect orientation to C programming but most of the work has been performed in parts focusing on single aspects.

The work done by Coady *et. al.* [28] depicted how AOP can be used to refactor *prefetching code* in the FreeBSD OS kernel. The new solution proposed in terms of AspectC depicted many significant benefits such as independent development of the prefetching modes and overall improved comprehensibility. Their work does not focus on a general approach for isolating crosscutting concerns, since they restructured the code manually in an ad-hoc way.

Prior to this study many more researchers had investigated the utility of applying AOP to various crosscutting concerns. One of the earliest studies was conducted by [29] for preparing the code for isolating concerns and performing the necessary restructurings and concluded that the aspect solution does reduce the code size.

Lippert's [27] dealt with exception detection and handling code in a large Java framework. Both works discuss advantages of using AOSD, such as reduced code duplication and improved cohesion, and discuss some particular limitations of using AspectJ. Bruntink et al. present their experiences of [25] [26], solving crosscutting concerns in embedded C code to using aspect oriented programming. They [26] developed a domain-specific language (DSL) for parameter checking.

Gradually, many other aspect languages [30] also evolved that worked on the similar line of bringing aspect oriented software development to the C programming language.

The evolution of aspect oriented concept can be a great benefit to the procedural legacy refactoring world especially in refactoring the cross cutting concerns that enhance code entanglement and increases code cohesion. For example logging or security is very often mentioned as an example of cross cutting concern that is scattered throughout the source code. Code entanglement and crosscutting concerns form a major part of procedural programming. The individual goal of both refactoring and AOP is creating systems that are easier to understand and maintain without requiring huge upfront design effort. A combination of the two aspect-oriented refactoring helps in reorganizing code corresponding to crosscutting concerns to further improve modularization and get rid of the usual symptoms of crosscutting: code-tangling and code-scattering.

In Fortran , the aspect orientation is still in the phase of gaining significance but as has been seen that fortran has also evolved from the traditional procedural language to object oriented fortran the code entanglement introduced through cross cutting concerns have not been focused in refactoring. Therefore, the target of Fortran refactoring was mostly to improve readability and performance. In C the exception handling that is solved using idiomatic concerns can be handled using exception handling methods with aspect oriented constructs. While rare today, some languages don't provide for effective *modularity*, especially the procedural languages which mean the breakup of code into units that can be maintained separately and used in at least two different composite programs. This inhibits refactoring of functionality common to domains or feature-sets into associated libraries. This is critical, but yet more modularity can come from eliminating *required cohesion*, where language semantics force that a piece of code be defined or shadowed all in one place (this advantage being related to Cross Cutting Concern).

In the world of Cobol programming, though refactoring may not be a common practice but code transformations and automated reengineering transformation are frequently done to migrate, renovate or integrate the Cobol code[24].But the transformation reengineering problems and definite AOP problems are separated from a poorly understood borderline. Thus there are number problems such as for web enabling, the dumb terminal I/O are to be replaced by CGI-based HTML pages, Active Server Pages or others[11] in which AOP does not serve the purpose. But as has been mentioned in the sections above, the typical crosscutting problem such as the logging and tracing concern cannot be tackled easily in classic Cobol. Another example such as error checking and error handling can be readily handled by aspects [11].

The significance of aspect based refactoring is established more in situations where existing concerns are to be updated, deleted or new concerns are to be added that relate to the scalability of the system as has been depicted in the case of C language in the above section. This is because the modifications required to achieve them may be very complex or may result in additional bug to the software.

VI. CONCLUSION

In this paper we have analyzed the work that is done in the field of refactoring the procedural languages. Our area of focus is specifically 3 languages that are Fortran , Cobol and C. We have highlighted the contributions that have been done in the area of refactoring these languages and have also extracted the shortcomings that exist in the existing refactoring techniques. As each of these languages has a different history, origin and application areas therefore their evolution process also differ significantly. Fortran that is designed specifically for scientific applications has a complex evolutionary process of transforming from structured procedural language to an object oriented version gaining a leading role in High Performance Computing world. The major refactorings focus on improving maintainability and performance that majorly deals with improving readability and removing poor Fortran coding practices. But the problem of crosscutting concerns still remains a difficult task to be solved in Fortran. Whereas, Cobol that still serves to be a business critical language has billions of lines of code in use worldwide. Though the Cobol literature does not focus much on refactoring but it has to undergo a number of reengineering transformations for the purpose of migration, renovation or integration. But world wide companies using Cobol are in strict need of complying with the new standards and transforming and refactoring the systems with aspects is being adopted these days for achieving better separation of concerns. Therefore, the role of aspect orientation is gradually increasing in reengineering the Cobol legacy systems also. The C programming language has widespread applications and a number of refactoring strategies have been adopted for code improvement. But even the traditional refactorings performed in C fail to achieve proper modularization. The study tries to extract the refactoring pattern that these languages follow and the patterns that may benefit these languages. Our approach to refactoring using aspect orientation in C can help the programmers gain insight to improve upon the development productivity and support for changes in the requirements in other procedural languages as well. With the acceptance of Aspect-oriented programming as a tool for the identification of concerns, it is gaining a bigger role in the refactoring scenario. Especially in tackling the concerns that are complex or impossible to handle using the traditional refactoring techniques, the aspect oriented constructs makes it easy to achieve.

REFRENCES

[1]M.Mendez, J.L. Overbey, A. Garrido, F.G.Tinetti and R.Johnson "A Catalog and Classification of Fortran Refactorings for Legacy Systems, 2011.

[2] J. L. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for Fortran and High-Performance Computing. In *SE-HPCS '05*: Proceedings of the second international workshop on Software engineering for high performance computing system applications, New York, NY, USA, 2005. ACM.

[3] J.L. Overbey, S. Negara, and R.E. Johnson. Refactoring and the Evolution of Fortran. In *2nd International Workshop on Software Engineering for Computational Science and Engineering (SECSE'09)*, 2009.

[4] Roberts, D., Brant, J., and Johnson, R. "A Refactoring Tool for Smalltalk." Theory and Practice of Object Systems 3(4), 1997.

[5] De, V. A Foundation for Refactoring Fortran 90 in Eclipse. M.S. Thesis, University of Illinois at Urbana-Champaign, 2004.

[6] A. Garrido. Software Refactoring Applied to C Programming Language. Master's thesis, University of Illinois, 2000.

[7] A. Garrido and R. Johnson. Program Refactoring in the Presence of Preprocessor Directives. *University of Illinois at Urbana-Champaign, Champaign, IL*, 2005.

[8] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[9] D. Spinellis. CScout: A refactoring browser for C. Science of Computer Programming 75 (2010) 216_231, 2009.Elsevier.

[10] Bodhuin, T. Guardabascio, E. Tortorella, M. Migrating COBOL systems to the Web by using the MVC design pattern,2003.

[11] Ralf Lamme and Kris De Schutter. What does aspect-oriented programming mean to Cobol? Proceedings of Aspect-Oriented Software Development (AOSD 2005).

[12] Jianjun Pu, Zhuopeng Zhang, Jian Kang, Yang Xu and Hongji Yang. Using Aspect Orientation in Understanding Legacy COBOL Code, 2007.

[13] Rizvi S.A.M and Khanam Z. Refactoring Catalog for Legacy software using C and Aspect Oriented Language. In the proceedings of SERP'11, WorldComp 2011, Las Vegas, USA.

[14] Monteiro M.J.T.P. Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts, 2005.

[15] Opdyke, W. Refactoring Object-Oriented Frameworks. PhD dissertation, University of Illinois at Urbana-Champaign, 1992.

[16] Mariano Mendez .Fortran Refactoring for Legacy Systems.2011

[17] Eric Brewer and Bill McCloskey, "ASTEC: a new approach to refactoring C",ACM SIGSOFT Software Engineering Notes, Volume 30 Issue 5,September 2005,Pages 21-30.

[18] Jingfeng Peng, "Semi-Automated Refactoring Applied to the C Programming Language", 2008.

[19] X-Ref. Xrefactory - A C/C++ Development Tool with Refactoring Browser, May 2007. http://www.xref-tech.com.

[20] M. Eaddy, T. Zimmerman, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho,"Do Crosscutting Concerns Cause Defects?," IEEE Transactions on Software Engineering , 2008.

[21] B.B.Boniati,A.S.charao,B.O. Stein,G.Rissetti,E.K.Piveta,May 2011, "Automated refactorings for High performance Fortran programmes",International Journal of High Performance Systems Architecture, Volume 3 Issue 2/3,May 2011,Pages 98-109.

[22] H. Shinomi, Y. Ichimori. (2010), "Program Analysis Environment for Writing Cobol Aspects",Japan.

[23] T. Morioka, H. Danno, and H. Shinomi. An aspect-oriented cobol for the industrial setting. In *7th International Conference on Aspect-Oriented Software Development (AOSD.08)*, Apr. 2008.

[24] N.P. Veerman. Revitalizing modifiability of legacy assets. *Journal of Software Maintenance and Evolution*, 16(4-5):219–254, July-Oct 2004. Special issue on CSMR 2003.

[25]M. Bruntink, A. van Deursen, and T. Tourw´e. Isolating Idiomatic Crosscutting Concerns. In Proceedings of the 21th International Conference on Software Maintenance (ICSM). IEEE Computer Society, 2005.

[26] M. Bruntink, A. van Deursen, and T. Tourw´e. Discovering Faults in Idiom based Exception Handling. In *the* Proceedings of the 28th international conference on Software engineering Pages 242-251 , 2006.

[27] M. Lippert and C. V. Lopes. A study on exception detecton and handling using aspect-oriented programming. In *Proceedings of the 22th international conference on Software engineering (ICSE)*, pages 418 – 427. IEEE Computer Society, 2000.

[28] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code. In *Proceedings of the Joint European Software Engineering Conference (ESEC)and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)*, pages 88–98. ACM Press, 2001.

[29] G. C. Murphy, A. Lai, R. J. Walker, and M. P. Robillard. Separating Features in Source Code: An Exploratory Study. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 275–284. IEEE Computer Society, 2001.

[30] B. Adams and T. Tourw´e. Aspect-Orientation in C: Express Yourself. In *Proceedings of the AOSD Workshop on Software-engineering Properties of Languages for Aspect Technologies (SPLAT).* Aarhus University, March 2005.

[31] K.D.Schutter , B.Adams. Aspect-orientation for revitalising legacy business software.ERCIM2006.