Application of AOP Methodology in Eclipse-AJDT Environment for Developing Bioinformatics Software

Amita Sharma
B-16, "Tulsi Shree"
Kanta Khaturia Colony
Bikaner – 334003, India

S.S. Sarangdevot
Director, Deptt. Of Computer Science & I.T.
J.R.N. Rajasthan Vidyapeeth (Deemed)
University, Udaipur – 313001, India

ABSTRACT
The application of Aspect-Oriented Programming (AOP) methodology has been investigated in the development of Bioinformatics software – Bioseqsearch. This software aims to reveal the biological significance of an unknown sequence using similarity search through biological databases using NCBI BLAST via internet. The complexity of the design has been significantly reduced by achieving better separation of concerns through modularization of identified crosscutting concerns, thus eliminating the problems of code scattering and tangling. The impact of using this methodology on various quality factors of the software has been examined. The study concludes that AOP methodology in Eclipse-AJDT environment is highly useful in design and implementation of efficient, cost-effective and quality bioinformatics software projects.

General Terms
Design, Experimentation, Languages, Theory, Verification.

Keywords
Aspect-Oriented Programming, Separation of Concerns, Bioinformatics Software, Eclipse-AJDT.

1. INTRODUCTION
Aspect-Oriented Programming (AOP) [1] [2] is an emerging programming paradigm that complements Object-Oriented Programming (OOP) [3] and enables the modular implementation of crosscutting concerns [4]. Invented by Kiczales [1], it aims at providing better means of addressing the well-known problem of separation of concerns [5] for reducing software complexity. AOP defines a new program construct – 'aspect' [6][7], which is a software entity that implements crosscutting functionality in a modular way. This provides most promising solution for elimination of code scattering [8] and tangling [9], thus overcoming the limitations of OOP.

In AOP the system is divided into two halves: the base (core) program and the aspect program. The base program contains the main functionality (core concerns) of the system and is implemented using OOP methodology. On the other hand, the aspect program consists of the crosscutting functionality (crosscutting concerns) and is implemented using AOP methodology. This facilitates better separation of those concerns that OOP handles poorly. Aspects are woven into the core program using an aspect weaver [8][9], thus the final software system is realized.

Kiczales and his team at Xerox PARC developed the first and most successful AOP language: AspectJ [10][11]. AspectJ is a general purpose AOP extension to Java. It adds to Java a few new constructs: poincuts, advice, intertype declarations and aspects. AspectJ's aspects work side by side with Java classes to develop a comprehensive application. Eclipse-AJDT [12]-[16] provides the most popular and commercially successful open source enhanced IDE support for AspectJ implementations, with a rich set of features like Aspect Visualizer, Outline View, Editor Support and Debugger.

Application of AOP design methodology using Eclipse-AJDT environment has previously been investigated in modular design of application software in the domain of banking [17] and insurance [18]. However, this methodology has not so far been investigated in the domain of bioinformatics.

Bioinformatics is a new scientific discipline at the crossroads of biology, medicine and information technology [19]. It involves collecting, manipulating, analyzing, and transmitting huge quantities of biological data and uses computers whenever appropriate [20]. The information archive in each organism is the genetic material DNA (deoxyribonucleic acid) [21]. DNA molecules are long linear chain molecules containing a message in a four – letter alphabet: A, C, G and T [22] and as such they are regarded as the brain of living cell [23]. A protein is a linear sequence of simpler molecules called amino acids. They are the molecules that accomplish most of the functions of a living cell [24], determining its shape and structure. Like the DNA, proteins are conveniently represented as strings of letters expressing the sequence of amino acids [25]. DNA and protein sequences are the main kinds of information stored in biological databases maintained at NCBI (National Center for Biotechnology Information), EMBL (European Molecular Biology Laboratory) and DDBJ (DNA Data Bank of Japan). The most powerful method of investigation of vast amount of biological data is comparison of bio-molecular sequences, because high sequence similarity usually implies significant functional and structural similarity.

In the present study, the application of AOP methodology has been investigated in the development of Bioinformatics Software – Bioseqsearch, using Eclipse-AJDT environment. Bioseqsearch aims to reveal the existence of similarity between an input sequence (query sequence) and other sequences (target sequences) stored in a biological database using NCBI BLAST server. Similarity search through biological databases using heuristic

algorithm of BLAST for local alignment, makes the task of deciphering the structure and biological function of a piece of DNA much easier. In this real-life working application, the complexity of the design has been significantly reduced by achieving better separation of concerns through modularization of identified crosscutting concerns, thus eliminating the problems of code scattering and tangling. Full benefits of AOP are realized using different visual and navigational features of Eclipse-AJDT environment. The impact of using this methodology on various quality factors of the software has been examined.

The rest of the paper is organized as follows: Section 2 describes the background of sequence databases and the requirements of the software for sequence similarity search. The problems associated with the traditional OOP design are stated and the solution followed in the present design is outlined. Section 3 presents an overview of our exemplary bioinformatics software - Bioseqsearch and Section 4 focuses on the identification of crosscutting concerns. Section 5 explains the identification of aspects. AOP design and implementation are presented in Section 6 and Section 7 discusses the observations regarding the impact of using AOP methodology in Eclipse-AJDT environment on various quality factors of the designed software. Section 8 provides the concluding remarks and Section 9 outlines the future work.

2. BACKGROUND

In the early 1960s, Dayhoff et al. [26] at the Protein Information Resource (PIR) collected all the protein sequences known at that time and published them as the Atlas of Protein Sequence and Structure. Later, text-based descriptions and information (annotations) regarding evolution of protein families were added to it. Soon this printed database became unwieldy and its electronic format became necessary. European Molecular Biology Laboratory (EMBL) initiated DNA sequence database in 1982. A few years later, GenBank and DDBJ joined the field. Collaboration among the three databanks started in 1988 and now, new or updated data are shared among them, once every 24 hours, in a flat file format [27]. Many other databases- primary, composite and secondary have also been developed and maintained.

These highly annotated databases are of great help to researchers. Very often, a researcher comes across a novel DNA or protein for which no functional data is available. He urgently needs some basic information about the sequence, before performing some meaningful experiment to decipher its biological function. Sequence similarity search through these databases comes to his rescue.

Basic Local Alignment Search Tool or BLAST [28][29] is the most widely used technique for detecting similarity between sequences of interest. It has become popular largely because it is very efficient and its algorithm is straightforward. It is able to detect similarities accurately between nucleotide or protein sequences quickly without sacrificing sensitivity.

Before submitting the query sequence to NCBI BLAST server, the sequence is to be converted to FASTA format. This requires a GUI, which should provide a window for entering the sequence in the text format. The software should automatically recognize the type of sequence (nucleotide or protein) and should convert it to FASTA format when the user clicks the format button. It should also allow the user to select suitable parameters for BLAST search. As the user clicks the submit button, the software should submit the sequence with user selected parameters to NCBI BLAST server via internet for similarity search. Once the search is complete, the software should receive, save and display the BLAST output, as desired by the user.

When such application software is designed using software engineering approaches and OOP methodology, the presence of crosscutting concerns impairs the modularity and quality of the software. The software becomes more and more complex due to code duplication (code scattering) and other problems (code tangling).

These problems have been solved in the present design of the bioinformatics software using AOP methodology, to complement OOP methodology. After identification of core concerns and crosscutting concerns of the system, core concerns have been implemented as classes using OOP methodology and crosscutting concerns have been implemented as aspects using AOP methodology. Classes and aspects are combined using weaving technique of AspectJ to deliver the final software system. Eclipse-AJDT provides the most advanced IDE for the software design and implementation.

3. BIOINFORMATICS SOFTWARE: Bioseqsearch

Bioseqsearch is application software which is designed to search the input query sequence using NCBI BLAST server. It takes user input (the sequence of nucleotide or amino acid), and visualizes the results of the database search (the BLAST output).

BLAST or the Basic Local Alignment Search Tool is a database search tool, developed and maintained by the NCBI – National Center for Biotechnology Information. The BLAST suit of programs has been designed to find high scoring local alignments between sequences, without compromising the speed of such searches. BLAST uses a heuristic algorithm which seeks local alignments and is able to detect relationship among sequences which share only isolated regions of similarity.

Bioseqsearch interface has a text area in which user provides input information to the application. The input information is a nucleotide or protein sequence. As the user clicks the "Format sequence" button present on the interface, the input information is read by the application.

The application automatically recognizes the sequence type (DNA, RNA, protein etc.), loads it in the Fasta format (as required by NCBI BLAST server) and presents the appropriate BLAST options. The invalid options are disabled. User selects the necessary BLAST parameters: the type of BLAST program, the database for searches, the E-value and then clicks the submit button present in the interface. The application thereafter sends the query sequence to the NCBI BLAST server with selected BLAST parameters for the search operation. After the completion of the search, the software receives, saves and displays the BLAST output as desired by the user.

Highly annotated records of BLAST output sequences provide useful information for ascertaining the structure and function of the input nucleotide/protein sequence.

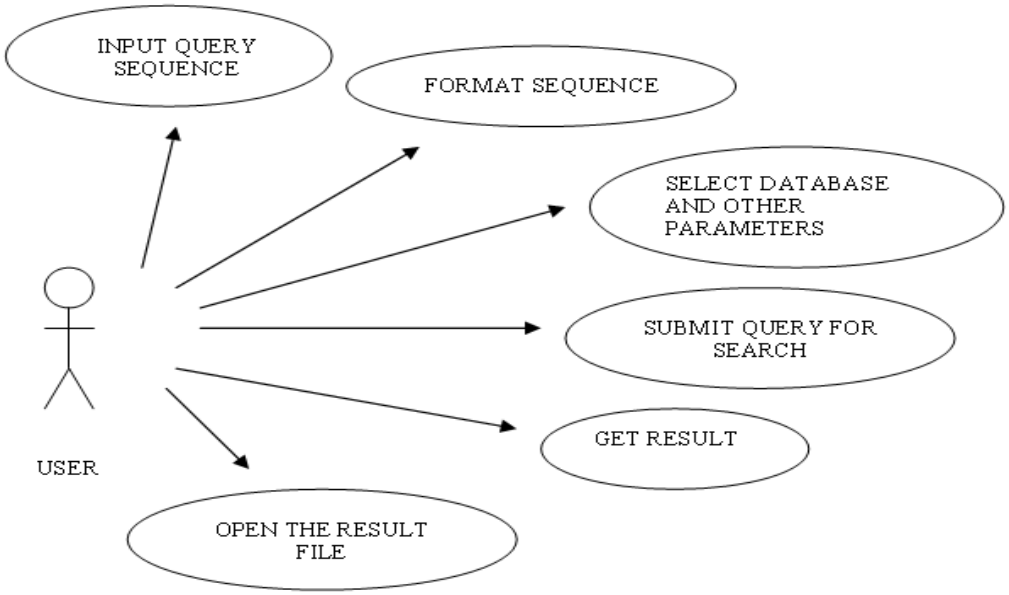

Figure 1. Use-Case Diagram of Bioseqsearch Software

4. IDENTIFICATION OF CROSSCUTTING CONCERNS

Separation of concerns and modularity are at the heart of the programming process. Being able to keep concerns separate is extremely important in software development. It helps in breaking down a complex problem into smaller parts, and solves them individually. When it comes to large systems, it is the only way to build them. Following this approach, the system is structured into units of function and behavior which can be put together to produce a complete software system.

First step in designing the bioinformatics software system is to develop a **Use – Case diagram.** A use-case is a typical interaction between a user and the system under development. Figure 1 shows the use-case diagram of the bioinformatics system. This diagram explains the requirements from the user's perspective.

From the study and analysis of the system, it is noticed that some functionalities cannot be localized into single modular units using OOP methodology. They represent functional crosscutting concerns: Nullreport, Validseq, Updatestatus and Notifyobserver. Apart from these, there are two more infrastructure concerns in the system, which are crosscutting: Logging and Tracing. Thus there are six crosscutting concerns in the bioinformatics system and they cannot be modularized using OOP methodology.

A successful solution to this problem involves two things: an engineering technique to separate such concerns from requirements all the way to code and a composition mechanism to merge design and implementation for each concern to result in the desired system. AOP methodology provides such a solution.

5. IDENTIFICATION OF ASPECTS

While examining the requirements of the system it was observed that some concerns are scattered over multiple classes and crosscut. If these concerns are not captured, they may cause code scattering and tangling. OOP is unable to capture them. In section 4, we have investigated and identified six such crosscutting concerns in this software system, which can be best represented as aspects for better separation of concerns using AOP methodology.

5.1 Nullreport Aspect

Whenever the application reads the sequence area, it checks whether the sequence text is null or not. Repeating this checking process at various locations of the program, where sequence text is read, will cause code scattering. This problem can be solved using AOP if we create Nullreport Aspect for this work. This aspect has a pointcut input_seq() and a join point String *.getText(). Using this aspect, if the input sequence read by the getText() method is null; the user is informed about it.

5.2 Validseq Aspect

While reading the user input sequence, it is necessary to check its validity. Placing checks for validity at various locations in the program will create code scattering and tangling. This problem can be avoided if Validseq Aspect is added. This aspect has two pointcuts: seqvalidity1() associated with join point String *.getText() and seqvalidity2() associated with join point updateSequenceArea(String, String, Boolean) method of SeqForm2 class. Using this aspect, whenever the sequence area is read or updated by the application, validity of the sequence is

checked. If the sequence is invalid, user is informed about it and the sequence area is cleared.

5.3 Updatestatus Aspect

There are various locations in this application where status is updated. Repeating calls for method statusaction() for status updating, will increase complexity of the program. Creating Updatestatus Aspect is a better choice which will be invoked as soon as the join point is executed. This aspect has four pointcuts. Updatestatus Aspect will invoke statusaction() method of SeqForm2 class and will update the status of the application.

5.4 NotifyObserver Aspect

In JQBlast class, at various locations, setChanged() method is executed and thereafter notifyObserver() method is called. To reduce code scattering we create Notifyobserver Aspect which is associated with join point java.util.Observable.setChanged() method and pointcut reportuser(JQBlast). This aspect will notify the user about the processing of application without code scattering.

5.5 Logging Aspect

Whenever the application submits the query sequence to NCBI BLAST server, the log file is maintained. The file helps in recording the result of query sequence with its BLAST parameters. For achieving this, without code scattering and tangling, Logging Aspect with pointcut Queryseq(JQBlast c, HashMap p) and join point JQBlast.submitQuery(HashMap) is added to the application. This aspect helps in graceful evolution of the application software.

5.6 Tracing Aspect

Every time the user clicks the button placed on the interface, the trace of action performed is made. Similarly, whenever the query result is received by the application, the trace of the join point is made. This helps in understanding the internal working of the application. For this purpose Tracing Aspect is added to the application.

6. AOP DESIGN AND IMPLEMENTATION

The bioinformatics software – Bioseqsearch has been developed using AOP methodology, to satisfy the requirements of the system. It is user friendly and menu driven. We have employed AspectJ as the AOP language in Eclipse-AJDT environment.

The Software includes three packages:

(i) **org.forms package:** has interface classes SeqForm2, Welcomeform.

(ii) **org.blast package:** Important classes are Blast, BlastException, BlastManager, JQBlast, RequestIdentifier etc. These classes connect the application to NCBI BLAST. They help in interchanging the information.

(iii) **org.aspectpack package:** All the aspects are stored in this package.

Crosscutting concerns have been modeled as aspects using the join point model of AspectJ. Each aspect has its well defined join points, pointcuts and advice. Figure 2 shows class diagram with functional aspects. Codes of two aspects are shown in APPENDIX I for illustration, while Figure 3 shows the sequence diagram of Updatestatus aspect. Aspects are woven in the main program by aspect weaver to produce the final system. This is done by the AspectJ compiler through AJDT. In this study, AspectJ Development Tools (AJDT) provided good tool support for editing, building and debugging AspectJ programs on Eclipse Platform. AJDT provided several aids to assist programmers in understanding AOP with AspectJ. The most important are Outline view, Cross References view, Aspect Visualizer and Debugger. For advice within an aspect, the Outline view shows the places in the program that will be affected by that advice. The links are navigable, so clicking on them opens an editor directly at the affected location. The Cross References view and the standard Outline view can be considered partners. Whereas the Outline view shows the structure of the current document, the Cross References view shows the crosscutting relationships for the current elements. Figure 4 shows the Cross References view placed below the corresponding Outline view for the relevant aspect. These views are essential programming aid and feedback tools that are used to verify that a piece of advice is matching in all the join points it was intended to match.

The most powerful tool AJDT provides for understanding the impact of aspects across the whole system is its Aspect Visualizer. Figure 5 shows the screen shot of Aspect Visualizer, which represents the classes and aspects within the application as bars and the places where aspects affect the code as stripes on the bars.

The lengths of the bars are relative to the file size – the longer the bar, more lines of code there are in the file that bar represents. With AJDT it is possible to step through the execution of advice in the debugger window and observe the full flow of the program to gain an understanding of the program's behavior. These features of AJDT help viewing how the application behaves and how different classes are affected by the aspects. This ensures correct implementation of aspects.

Bioseqsearch software is menu driven. It has several views and forms. Welcome screen is the entry screen. Application Service screen provides various services like format sequence, selection of program name, database name and E-value. These values are submitted to NCBI BLAST for similarity search by selecting submit button. The searched results are displayed by console view. Saved results can be opened and viewed. Figure 6 shows screen shot of BLAST output display for a given GenBank ID input.

For verification and validation, the software was tested by entering different pieces of known sequences and the searched outputs were examined. It was found that all the functionalities of the system worked correctly. This indicated the correctness of the implementation and the software met the desired specifications and needs.

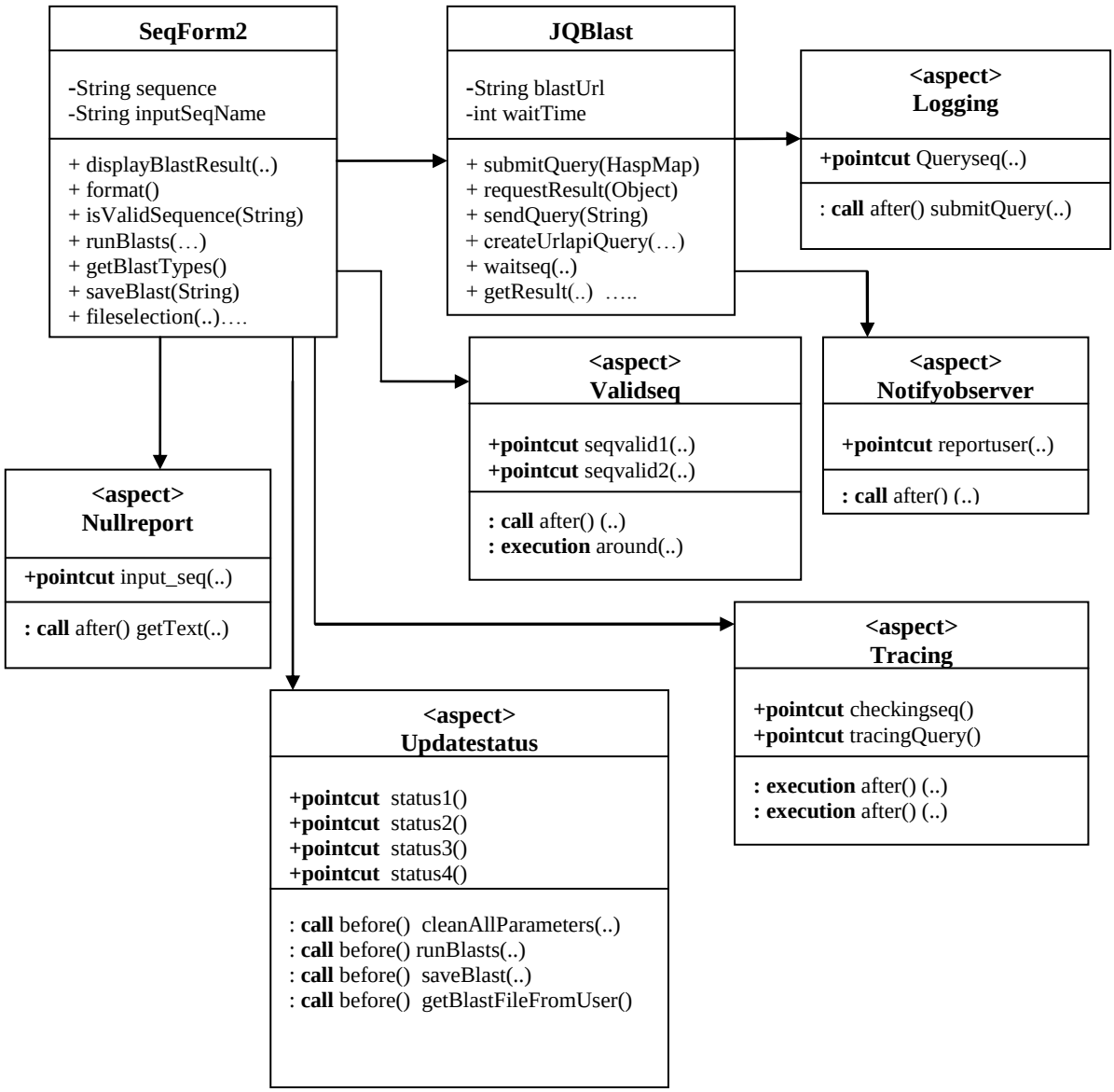

Figure 2. Class Diagram with functional aspects

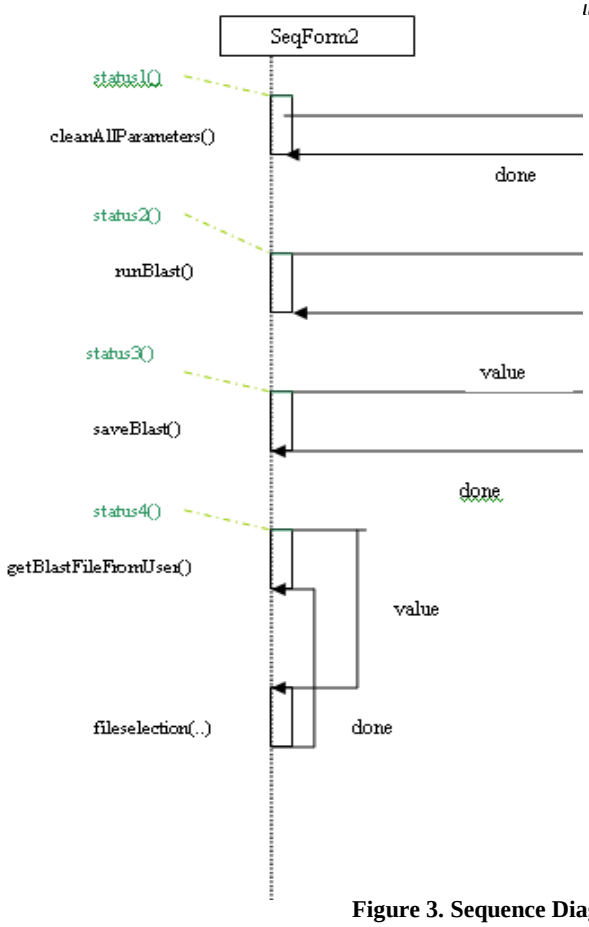

Figure 3. Sequence Diagram of Updatestatus aspect.

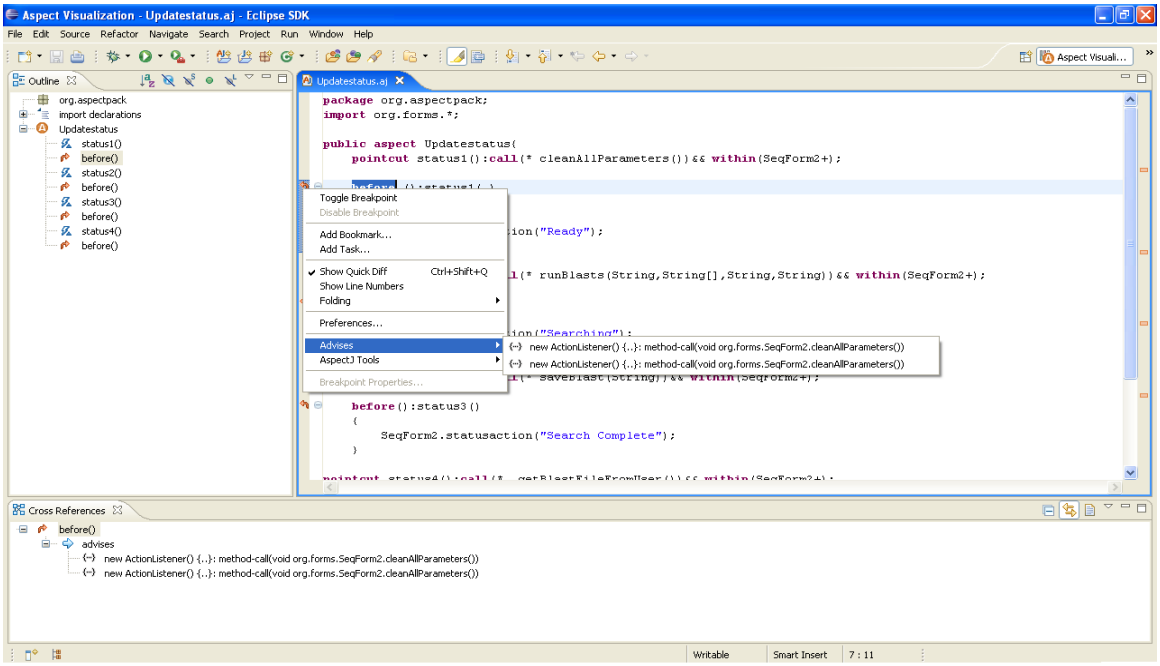

Figure 4. Outline, Cross References and Advice view of Updatestatus aspect

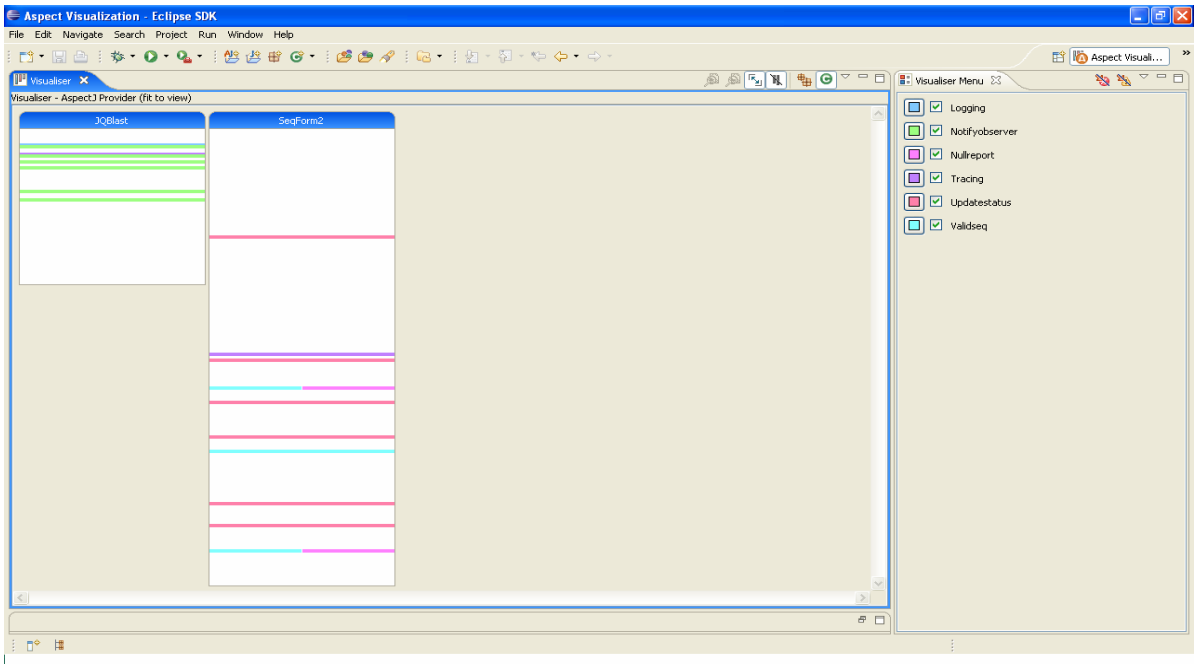

Figure 5. Aspect Visualiser view of Bioseqsearch Application

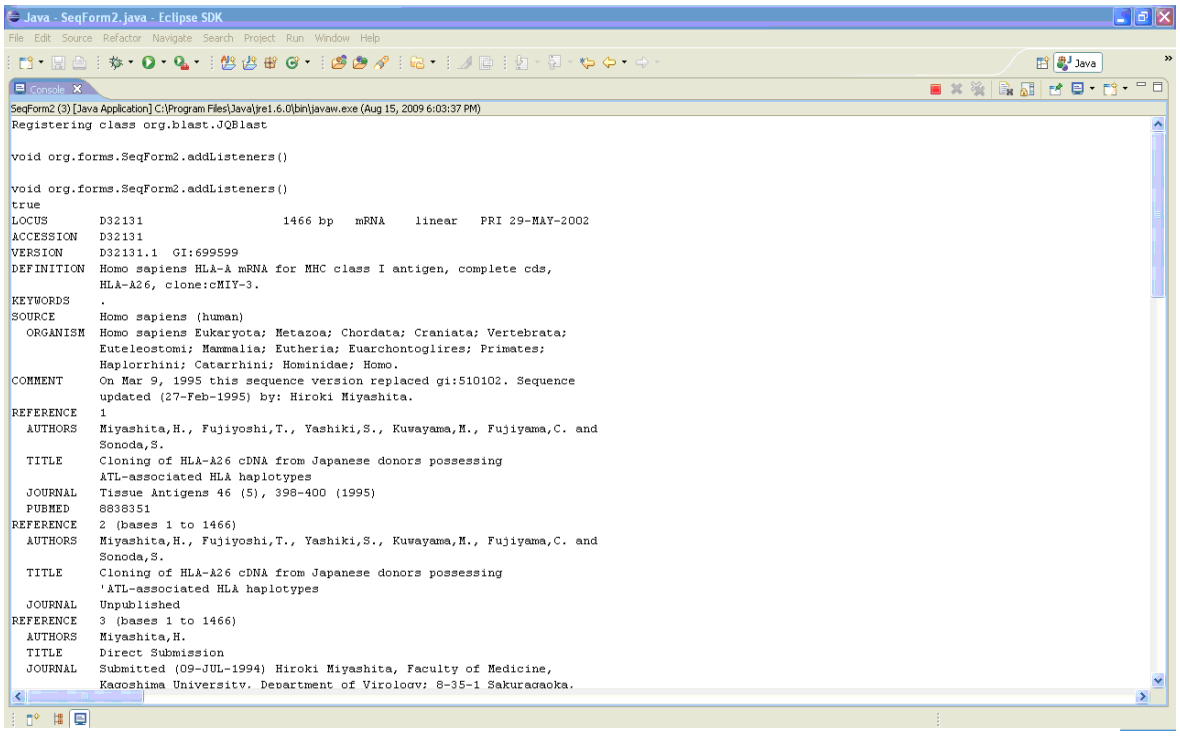

Figure 6. Console view of BLAST output display for a given GenbankID input

7. DISCUSSION

This investigation has been made to study the usefulness of the AOP methodology in real world bioinformatics applications. Impact of using this methodology on various quality factors has been examined. Observations made in the study are presented in this section.

Crosscutting concerns are an inherent part of all practical systems. Programmers are forced to write tangled and scattered code whenever they attempt to implement crosscutting concerns. This increases complexity of the design, degrades program's comprehensibility and decreases modularity, reusability, maintainability and customizability. In the present study, encapsulating the crosscutting concerns as aspects using AOP methodology, eliminated code scattering and tangling. In this way, *concerns were cleanly separated and modularity was enhanced*. This is in accordance with the observations of Bernardi et al. [30]. Enhancement in modularity *reduced the complexity and resources spent* in software development. This allowed a development team to assign experts to specific jobs, thus benefiting from their skill and experience. Cleaner separation of concerns and reduction in tangling, greatly *improved readability of the code*. This also resulted in *easier comprehension and better understandability*. It is in accordance with the opinion of Parnas [31] that modularization improves comprehensibility and reduces complexity.

In the system, there was cleaner assignment of responsibilities of different modules. This led to *improved traceability*. Using aspects it was always possible to add new functionality to the software without modifying the base program. This *enhanced* the *extendibility* of the software system.

The problem of software maintenance is widely known in software industry. By some estimates, 50-90% of software development resources are spent on software maintenance. Spiral model of Boehm [32] accepts it as part of software evolution. Thus producing software that is easy to maintain may potentially save large costs. In the current design, use of aspects *increased maintainability* of the program, because it allowed changes to particular functionality to happen in only one location. AOP design produces stand alone modules that can be changed whenever needed. Thus it is *easier to adapt* the software to accommodate the changing requirements.

Cohesive modular design in which each module implemented single concern only, *reusability* of the modules was *considerably enhanced*. It is in agreement with the findings of Elrad et al. [6]. Results of Maghawry and Dawood [33] also support the above conclusions.

With aspects there are more ways to code the same thing. This offers *more flexibility*. It was also noticed that with AOP design, less code was needed to be written which facilitated *easier system evolution* and reduced *development time.* The end effect was *cheaper implementation and better management of resources*. Reduction in duplicated code led to *clearer structure and less error prone implementation*. All these benefits make AOP a *favorite choice for trustworthy computing* [34].

However, there are a few limitations of AOP relevant to AspectJ:

1. Debugging an aspect is very tedious. Aspects are notoriously buggy during evolution because of a lack of useful error messages and warnings.
2. Generality of the advice is sometimes problematic because it may generate misleading error messages in a large system.
3. The capabilities of privileged aspects are far reaching because they can alter private member variables of any object within any class. This is very dangerous.

Thus AOP methodology needs to be applied with utmost caution and discipline.

8. CONCLUSION

In this study, a user friendly and menu driven application software for the selected bioinformatics software system has been designed and implemented using AOP methodology in Eclipse-AJDT environment. A good design is intended to be modular, and this is achieved through separating the crosscutting concerns. In this investigation, six crosscutting concerns were identified and they were modularized as aspects in highly cohesive modular units. Limitations of OOP design were overcome and the complexity of the design was considerably reduced due to elimination of code scattering and tangling. This application convinced that separation of concerns is one of the main requirements of good system design and implementation. Several benefits are attributed to software with well separated concerns.

Use of AOP methodology increased several software quality factors such as modularity, readability, understandability, maintainability, extendibility, reusability, correctness, traceability, flexibility, adaptability and ease of evolution. Reduction in development time and costing were also perceived.

Thus overall improvements in the quality and performance of the software system were realized. Several innovative visual and navigational features of Eclipse-AJDT environment made the development work of the software easy and reliable. Thus Eclipse-AJDT is the most suitable environment for developing aspect-oriented software. However, AOP methodology should be applied with utmost caution and discipline.

Successful implementation of the application concludes that AOP methodology in Eclipse-AJDT environment is highly useful in design and implementation of efficient, cost-effective and quality bioinformatics software projects.

9. FUTURE WORK

In the coming years, it is expected that the amount of sequence data generated around the world will outstrip that generated in the past decade. It may be in different structured formats on web [35]. It is very likely that the potential solution to the huge sequence data handling/storage problem will be the use of cloud computing. In this approach, a user will rent processing time on a computer cluster through a virtual operating system (or 'cloud'), which would load software and provide access point for running highly parallelized tasks. Sequencing data will be sent to the cluster either by disk or the internet. Thus, for next-generation sequencing, more effort must focus on developing software compatible for use in a cloud [36] [37] [38].

10. ACKNOWLEDGMENTS

The authors thank Prof. Divya Prabha Nagar, Vice Chancellor, J.R.N. Rajasthan Vidyapeeth (Deemed) University, Udaipur for encouragement and providing necessary research facilities.

11. REFERENCES

[1] Kiczales, G. et al. 1997. Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP) (Finland, June 1997), LNCS 1241, Springer-Verlag, 220-242.

[2] Elrad, T., Filman, R.E. and Bader, A. 2001. Aspect-Oriented Programming, Communications of the ACM 44, 10 (October 2001), 29-32.

[3] Sommerville, I., 2009. Software Engineering, 8th Edition, Pearson Education Limited.

[4] Kaur, A. and Johari, K. 2009. Identification of Crosscutting Concerns: A Survey. International Journal of Engineering Science and Technology 1, 3 (2009), 166-172.

[5] Hursch, W. and Lopes, C.V. 1995. Separation of Concerns. Technical Report NU-CCS-5-03 (February 1995).

[6] Elrad, T., Moderator: Aksit, M., Kiczales, G. Lieberherr, K. and Ossher, H. 2001. Discussing Aspects of AOP. Communications of the ACM 44, 10 (October 2001), 33-38.

[7] Apel, S. 2007. The Role of Features and Aspects in Software Development, Ph.D. Thesis, University of Magdeberg (2007).

[8] Gradecki, J. and Lesiecki, N. 2003. Mastering AspectJ, Wiley Publishing Inc.

[9] Laddad, R. 2003. AspectJ in Action, Manning Publication Co., Greenwich, CT.

[10] Kiczales, G. et al. 2001. Getting Started with AspectJ. Communications of the ACM 44, 10 (October 2001), 59-65.

[11] Kiczales, G. et al. 2001. An Overview of AspectJ, In Proceedings of ECOOP 2001 (Budapest, June 2001), LNCS 2072, 327-353.

[12] Colyer, A., Clements, A., Harley, G. and Webster, M. 2004. Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the AspectJ Development Tools, Addison Wesley Professional.

[13] AJDT: Frequently Asked Questions. http://www.eclipse.org/ajdt/faq.php

[14] Eclipse Platform Overview, The Eclipse Foundation, http://www.eclipse.org/whitepapers/eclipse-platform-whitepaper.pdf

[15] Colyer, A. and Clement, A. 2005. Aspect-Oriented Programming with AspectJ, IBM Systems Journal, 44, 2 (2005).

[16] Sharma, A. and Sarangdevot, S.S., 2010. Eclipse-AJDT: A Diamond from Open Source Technology, In Proc. Int'l Conf. Next Generation Communication and Computing Systems, Dec. 25-26, 2010, Chandigarh, India, 120-125.

[17] Sarangdevot, S.S. and Sharma, A. 2008-09. Investigating the application of AOP methodology in development of Banking Application Software Using Eclipse-AJDT environment. Journal of Management Sciences, AIMS, Udaipur, Rajasthan (2008-09), 124-141.

[18] Sharma, A. and Sarangdevot, S.S., 2010. Investigating the Application of AOP Methodology in Development of Insurance Application Software Using Eclipse-AJDT Environment, In Proc. Int'l Conf. Computer Engineering and Technology (ICCT '10), Nov. 13-14, 2010, Jodhpur, India, D17-D25.

[19] Bal, H. and Hujol, J. 2007. Java for Bioinformatics and Biomedical Applications, Springer.

[20] Bergeron, B. 2007. Bioinformatics Computing, Pearson-Prentice Hall.

[21] Lesk, A.M. 2004. Introduction to Bioinformatics, Oxford.

[22] Sergio Anibal de Carvalho Jr. 2003. Sequence Alignment Algorithms, M.Sc. Thesis, King's College, University of London.

[23] Wong, L. (Ed.), 2004. The Practical Bioinformatician, World Scientific Publishing Co. Pte. Ltd., Singapore.

[24] Atwood T. K. and Parry-Smith, D.J. 1999. Introduction to Bioinformatics, Pearson.

[25] Bourgaze, D., Jewell, T.R. and Buiser, R. 2004. Biotechnology: Demystifying the Concepts, Pearson.

[26] Dayhoff, M. O. 1978. Atlas of Protein Sequence and Structure, vol. 5, Suppl. 3, National Biomedical Research Foundation, Washington, DC.

[27] Baxevanis, A. D. and Ouellette, B. F. F. (Eds.) 2009. Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins, Wiley-India.

[28] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J. 1991. Basic Local Alignment Search Tool, J. Mol. Biol. 215, 403-410.

[29] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D. J. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucl. Acids Res. 25, 3389-3402.

[30] Bernardi, M. L. and Lucca, G. A. D., Improving Design Patterns Modularity Using Aspect Orientation, RCOST, University of Saunio, Italy.

[31] Parnas, D. L. 1972. On the Criteria To Be Used in Decomposing Systems into Modules, Communications of the ACM, 15, 12, 1053-1058.

[32] Boehm, B. 1986. A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, ACM, 11, 4 (August, 1986), 14-24.

[33] Maghawry, N. E. and Dawood, A. R., "Aspect-Oriented GoF Design Patterns," http://infos2010.fci.cu.edu.eg/uploadCamera2010/noura.elmaghawry/AO_Gof_Patterns.pdf

[34] Safonov, V. O. 2008. Using Aspect-Oriented Programming for Trustworthy Software Development, John Wiley & Sons, Inc., New Jersey.

[35] Cafarella, M.J. 2011. Structured Data on the Web, Communications of the ACM, 54 (2), February, 2011, 72-79.

[36] Editorial, Gathering clouds and a sequencing storm, Nature Biotechnology, 28, 1, 2010.

[37] Sansom, C., 2010. Up in a cloud?, Nature Biotechnology, 28, 13-15 (2010).

[38] O'Conner, B.D. et al., 2010. SeqWare Query Engine : Storing and searching sequence data in the cloud, BMC Bioinformatics 2010, 11(Suppl 12) : S2 (21 Dec. 2010).

APPENDIX I

```
pointcut ():call(* cleanAllParameters())&& within(SeqForm2+);

before ():status1( )
        {
                SeqForm2.statusaction("Ready");
        }
```

```
pointcut ():call(* runBlasts(String,String[],String,String))&& within(SeqForm2+);

before ():status2( )
        {
                SeqForm2.statusaction("Searching");
        }
```

```
pointcut ():call(*saveBlast(String))&& within(SeqForm2+);

before ():status3( )
        {
                SeqForm2.statusaction("Search Complete");
        }
```

```
pointcut ():call(*  getBlastFileFromUser())&& within(SeqForm2+);

before ():status3( )
        {
                SeqForm2.statusaction("Opening File");
        }
```

Code1. Updatestatus Aspect

```
pointcut Queryseq(JQBlast c,HashMap p):target(c)&&args(p)&&execution(Object JQBlast.submitQuery(HashMap));

after (JQBlast c,HashMap p):Queryseq(c,p)
        {
                System.out.println("logger history");
                String s="Query:";
                 String a=String.valueOf(p);
                s=s.concat(a);
                s=s.concat("\n");
                byte j[]=s.getBytes();
                try
                {                                       ff(j);
                } catch (IOException e) {System.out.println("error");  }
        }
```

Code2. Logging Aspect