Improving Agile Software Development using eXtreme AOCE and Aspect-Oriented CVS

Santokh Singh, Hsiao-Cheng Chen, Oliver Hunter, John Grundy, John Hosking
Computer Science Dept, University of Auckland, Private Bag 92019, Auckland, New Zealand
{santokh@cs, chsi022@ec, ohun002@ec, john-g@cs, john@cs}.auckland.ac.nz

Abstract

Currently there are no Concurrent Versioning Systems (CVS) designed to properly support agile software development. The existing CVS lacks user friendliness and it requires users to be fully experienced with the system before they can adequately use it. Also its asynchronous style of merging often leads to code loss. In this paper, we describe a novel CVS system, called the Aspect-Oriented CVS (AOCVS) and our newly derived agile software development methodology, eXtreme Aspect-Oriented Component Engineering (eXtreme AOCE). Unlike the general CVS which is used for a vast variety of projects, AOCVS was designed and developed specifically for eXtreme AOCE. Our CVS tool merges the changes in a synchronous/real time fashion; this in turn removes the hassle for the developers on having to resolve merging conflicts. We also describe how our implemented AOCVS can be used to provide functionalities that can assist developers to produce and distribute reusable code and to communicate and manage aspect-oriented agile projects more efficiently and effectively. Our novel agile approach and tool allows software developers to use the rich cross-cutting systemic concerns, their behaviour and properties in aspect-oriented components to refactor, maintain, add functionalities and test complex software systems more easily, rapidly and accurately.

1 Introduction

For a software project to be successful, it is vital and crucial that an efficient and appropriate methodology be employed and the right tools used [21, 9]. The particular methodology used determines the efficiency of the implementation process as well as the quality and maintainability of the final product [2, 9, 21], while the tools employed affects the effectiveness of the developers concerned [20]. This paper provides the formal description of the newly derived and novel methodology called eXtreme AOCE and our newly developed tool, called the Aspect-Oriented Concurrent Versioning System, which we designed and developed to support development using this methodology.

In the agile software engineering world eXtreme Programming (XP) is thought to be the most well known methodology [22]. Kent Beck [2] defined XP as a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements. XP values communication and simplicity of implementation [2, 9]. This methodology has the reputation to drive a team to produce high quality and functioning code that meets the requirement of the customers [2, 9, 13].

Compared to XP, Aspect-Oriented Component Engineering (AOCE) [3, 10] is a relatively new methodology and it focuses on building reusable software systems with aspect-oriented components. AOCE is a software development methodology that can be used from inception to design, development, deployment and subsequent maintenance. It constructs components that use the rich cross-cutting concerns called aspects as the building blocks of our software. Examples of aspects include those for persistency, security, performance and transaction processing [3, 10, 12, 15]. Aspectising components make them better characterised and categorised, thereby producing highly reusable, scalable, maintainable and understandable software.

No matter which methodology is used on a project, a software development team often requires some degrees of support for collaboration of work [7, 11, 16]. Collaboration can be divided into two types, synchronous and asynchronous collaboration [7, 11, 18]. Synchronous development is where multiple developers work on the same document at the same time [7, 18, 20] and asynchronous development is where groups of users take a document, work on that document and then try to merge their changes in a later stage [4, 14, 19].

Concurrent Versioning System (CVS) is a tool that is widely used to help projects to be collaborated asynchronously [4, 14, 19]. CVS keeps history of all versions of the project at each point in the development. The users of CVS are able to upload or commit their files to the CVS repository and also to download or update the files onto their local computer [4, 14].

In the following sections we will explain the motivation behind our work. We will then describe and discuss in-depth the novel methodology that builds and improves on traditional agile methods. We will also

describe and discuss a new tool to support software development using this methodology.

2 Motivation

XP is a powerful and effective agile methodology that is adaptable to project requirement change and it focuses on producing systems that provide and only provide the functions that are required by the customer [2, 9, 13]. However, XP does not specifically have support for code reusability and software cross-cutting concerns, and the larger and more complex the software system becomes the more critical these issues become.

AOCE on the other hand focuses on producing quality and highly reusable components and it resolves the cross-cutting problems by identifying, isolating and utilising the aspects within the software components [1, 3]. However, AOCE on its own does not provide any mechanism to support team management, communication issues and coping with change.

Upon weighing the strengths and weaknesses of both methodologies, we combined and further extended their concepts; including using one methodology's strengths to counter the weaknesses in the other. We called this new methodology eXtreme AOCE and it is extremely useful and versatile.

eXtreme AOCE extends and uses all the features of XP and has added support for the cross-cutting concerns which are lacking in XP [1, 3]. This new methodology will now drive a software development team to produce more understandable, maintainable, scalable and reusable code; and at the same time, eXtreme AOCE will increase the business value of a software development team by increasing reusable components in their development library [1, 3, 17].

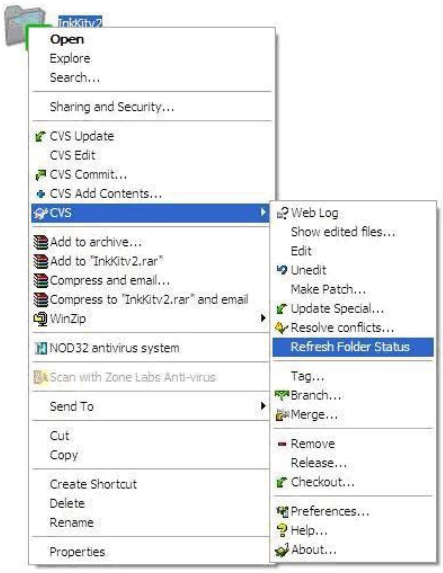

Figure 1 – Screenshot of a CVS

After the formulation and initial testing of eXtreme AOCE, we realised that we needed to create a better CVS system, which we called Aspect-Oriented CVS or AOCVS, to support this methodology. Having experienced CVS as a tool for keeping versions on software projects, we found that it is easy to make mistakes when using the system. Often these mistakes are due to the fact that CVS does not have a user friendly interface. A screenshot of the unfriendly and unhelpful CVS user interface is shown in figure 1 above, and further it does not have an easy to understand procedure of use. To compound to these problems, many critical issues arise at the merging stage, e.g. overwriting methods that were modified by someone else thereby losing valuable information, as CVS requires all users to be highly experienced with it first in order to be able to just use it.

The idea of AOCVS was as such conceived through necessity as every project requires the right tool that can help make the development process more effective and efficient, but the CVS just couldn't support this for eXtreme AOCE. AOCVS is designed to be an alternative to CVS and to provide specifically for the software development of projects done under the eXtreme AOCE environment. With AOCVS we aim to provide functions to remove the problems encountered when merging changes, and at the same time support cross-cutting issues and provide additional support for project management, also AOCVS will present eXtreme AOCE engineers a more user friendly and comprehensible interface.

3 eXtreme AOCE

This newly formulated methodology combines and extends both XP and AOCE concepts. eXtreme AOCE recognises and identifies the weakness in both the methodologies and addresses them. The following sub-sections describe the eXtreme AOCE principles and its practice.

3.1 eXtreme AOCE principles

eXtreme AOCE extends the 12 principles of XP and modifies them to incorporate the characteristics of AOCE so that it becomes more comprehensive and efficient to use when developing large and complex software systems. These principles are defined and described below.

The first principle called the Aspect-Oriented Planning Game, dictates that customers will carry out the planning game with the development team as they would in an XP environment [2, 9, 13], except that all planning is done by taking aspects and their cross-cutting properties into consideration as done in AOCE. Another difference is that during the estimation phase, the developers will be required to break down the stories given to them into smaller *aspect-oriented* stories which essentially dictate single aspect-oriented components. For example, the story "A user may

register with the system and then login" will be broken down by the developers into "A Security User Authenticate component for register and login" and "A Persistency Database access component for storing and retrieving data on user information". Developers will then estimate the time required for each of the broken down aspect-oriented stories and their sum total will be the estimated time for the story from the customer.

We use this story decomposition technique to improve the developer's accuracy on cost estimation and at the same time, provide the development team with the fundamental ideas about what aspect-oriented components need to be implemented.

The next principle is termed Small Aspect-Oriented Releases, and is in line with the XP principles [2, 9, 13], but each release will consists of a number of iterations that also consider the aspects and their properties. The release will display the features, chosen by the customer, provided by the implemented components that were constructed in the iterations.

The third principle is called the Aspect-Oriented Componentised System Metaphor which extends the system metaphor principle from XP which demands that developers use good naming convention [2, 9]. eXtreme AOCE further extends and reinforces this principle by introducing the notion of components richly cross-cut with aspects. With the use of aspects a method can be better categorized and characterized, and with the use of components, related operations can be better grouped together and located.

The fourth principle, Simple Aspect-Oriented Components Design states that each component should be designed with minimum functions required to satisfy the stories in the current iteration. Interface methods that are relevant to the definition of the component but not required by the current iteration maybe added to the interface and must be commented out. The underlying component classes must not implement the commented out interface methods.

The fifth principle is named Continuos Aspect and Components Testing, here unit tests are first created at aspect level; each method published by a component must pass the tests that were written for them. Every time changes are made to a component the tests written for that component will need to be rerun. The hierarchy of tests listing from the bottom up are aspect testing, component testing, component integration testing and customer acceptance testing.

The sixth principle is called Aspect-Oriented Components Refactoring. Refactoring is done at both aspectual and component level. eXtreme AOCE demands backward compatible refactoring, this means the signature of the methods already published in the component's interface should not be modified even if a method with similar function is to be created. The old method should only be commented as "out dated" rather than be deleted entirely.

The seventh principle, called Aspect-Oriented Pair Programming, demands that each pair working under the eXtreme AOCE environment focus their tasks at aspects level. Each pair will only be required to work on one method at a time thus allowing specific pairs to focus and specialise on certain aspects. For example, pairs familiar with database will be allowed to focus on Persistency aspects while pairs who are familiar with graphics can focus on User Interface aspect.

The eighth principle is Collective Aspect-Oriented Code Ownership. No matter what type of aspect a pair specialises in, they do not have the ownership of the code, the development team owns the code. This allows any pair to change any components and the methods within the components as they see fit.

The ninth principle is Continuous Aspect-Oriented Components Integration. Integration of code is performed whenever a component has been modified, with aspect-oriented concerns weaved in if required. When a method within a component is changed the code should be integrated. The tenth principle lays out the 40-Hour AOCE Work Week concept. All developers working under the eXtreme AOCE environment are expected to stick to the 40 hours a week work load. The eleventh principle is regarding the On-site Customer, here the development team will include one or more members who represent the customer and they will dictate what features the product should or should not have.

The last principle, Aspect-Oriented Components Coding Standards, lays down the aspect-oriented coding style to adhere to comment and name the components, methods and aspects which the development team itself will need to decide and agree upon. It is very important to define the purpose and function of a component the moment it is created. This will increase the understandability and role of the aspect-oriented component to any other eXtreme AOCE pairs not directly involved in its construction.

3.2 The practice of eXtreme AOCE

Communication is the key to getting the most out of eXtreme AOCE. When a pair receives a broken down aspectised story, they must consult with other pairs to check if there is a component which caters to the requirement of the story. If such a component does already exist, the pair will need to test and determine if the component needs to be modified or extended to satisfy the story. If no such component can be found, then a new component will be designed and constructed. Thus communication is essential in eXtreme AOCE as this will eliminate the risk of different pairs reinventing the wheel by duplicating work already done. This will save both time and costs besides effort, thus increasing productivity and efficiency.

4 Aspect-oriented CVS

Our novel AOCVS had been developed to be a system that will improve and streamline software project development carried out using the eXtreme

AOCE methodology. It is a CVS that not only provides the general CVS functions but also allows group communication, management, synchronous merging and improved eXtreme AOCE project support. AOCVS has features that reduce the coding time and has the synchronous merge function that solves some of the original CVS problems that we discussed earlier.

AOCVS is a variation and extension of CVS, therefore it also provides all the basic function that a CVS supports. Since the target user group of AOCVS are software developers who practise eXtreme AOCE, AOCVS provides additional functions to support the use of this methodology and for its software implementation through agile techniques. All these functions are described in the following subsections.

4.1 Basic functions

Just like a normal CVS, AOCVS allows a user to upload or download files, commit changes, login and logout, but it also has functions such as for search and chat. The search function is for finding required aspects, methods and keywords within the group's files. The chat function that is built-in helps developers and users to communicate instantly within the whole group or between smaller groups. We even tested this function for instant communication within the members of a particular pair doing eXtreme AOCE programming, and discovered that the members of the pair need not be seated side by side as the instant chat function allows for instant communication and code access. As such, distance in not a barrier when developers use our AOCVS and the members of each pair may be separated by any amount of space, as long as there is web access between the two members they can do pair programming.

4.2 Synchronous merge

Synchronous merging is the primary way that code is merged within AOCVS. This is done by sending each character typed in the section editing window to the server which in turn broadcasts the change to each of the clients that have a window open on that particular section.

Synchronous merge was an idea obtained from the features of other tools [18, 20] and it is incorporated into AOCVS to address the drawbacks of the original CVS caused by asynchronous merge. Asynchronous merge can create conflict between two merging versions of a file. Our synchronous merging function solves this by allowing only one user to change a section at a time, thereby enabling all changes to be made automatically in real-time.

4.3 Section ownership

Each section in a file can only be edited by the developer who has ownership on that section. The users without ownership can only open the section as read only copies.

A user with ownership has the power to transfer his or her ownership onto another user, or simply release that ownership. This is designed to simulate the eXtreme AOCE coding environment whereby developers work in pairs with one having the control of the keyboard, the transfer of ownership simulates the switch control of the keyboard between the pairs. This design also allows developers to work in eXtreme AOCE fashion even though they may be separated by being in different locations.

4.4 Filtering function

Filtering functions, shown in figure 2 are provided to allow the users to filter out undesired aspects, their details and components from the project structure view.

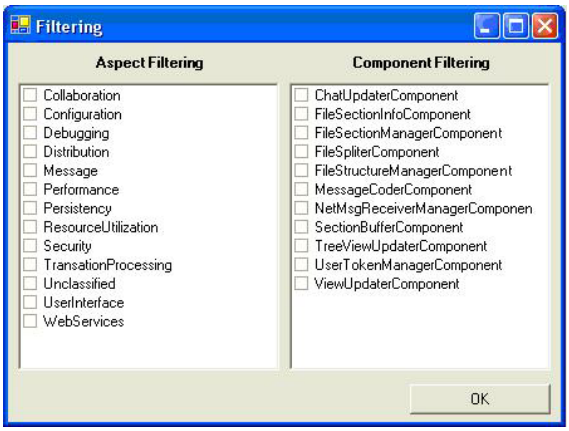

Figure 2 - Aspect and component filtering window

Filtering functions help software developers by only showing them what they need and want to see. For example, if a developer is working on database code then this user may only wish to view sections with persistency aspects. This developer would un-tick all aspects except persistency thus showing only the relevant code and filtering everything else out.

As an eXtreme AOCE developer, this is very useful because it clearly shows only the relevant aspects and components. It can help speed up the process of development due to not having to search through irrelevant code. eXtreme AOCE developers would also feel more comfortable because of the reduced clutter on the screen.

4.5 Cross-cutting concerns

AOCVS allows the user to reference an existing section of a file into another file. If a section is referenced into more than one file, then updating any of those sections will result in updating all the referenced sections in all the files.

When a section is opened it will be displayed with additional cross-cutting information, part of that

information lists the files which had referenced that particular section. This feature provides developers with additional useful information that can aid them on deciding the course of action to take as regards multi-referenced sections.

5 Implementation of AOCVS

The AOCVS is a tool we designed and developed using eXtreme AOCE for supporting the eXtreme AOCE methodology. We used .NET and Microsoft's technologies to implement our AOCVS system as we were more familiar with these compared to Java and J2EE. Also Visual Studio .NET allows for rapid prototyping and has rich support for the libraries that we needed. During the initial stage of AOCVS's development, only the AOCE methodology was used, the intention was to build the system with advanced code re-usability. However, due to the frequent specification changes, complication of duplicated code and time limitation, the methodology of eXtreme AOCE was adopted to enhance the efficiency of system's development.

5.1 Architecture of AOCVS

In following the principle of eXtreme AOCE, the AOCVS is constructed with independent and well defined components. This makes the AOCVS software highly re-usable and also allows for any future maintenance and extension on the system to be done with ease.

Each component within the system was designed to perform a specific task with limited coupling with other components. This setup allows the AOCVS to be refactored more easily and also made the components within the system to be more replaceable.

The AOCVS architecture shown in figure 3 can be identified as composed of three main partitions, namely the client, server and database. Each of these three partitions has a vital entry point component, called the NetMsgReceiver, AOCVS Webservice and DBManager for the client, server and database respectively. These components publish a list of functions which the partition is capable of providing. By having such components it further reduces the coupling within the system and makes the system more maintainable.

The maintainability of the system was apparent during the course of the development stage. For instance, the database consumed by the AOCVS was switched from Microsoft Access to Microsoft SQL, and later the structure within the database was overhauled and redesigned completely, making major changes to the AOCVS. But these changes only caused a minor delay in the development process, giving credibility to the comprehensiveness, efficiency and effectiveness of developing with eXtreme AOCE.

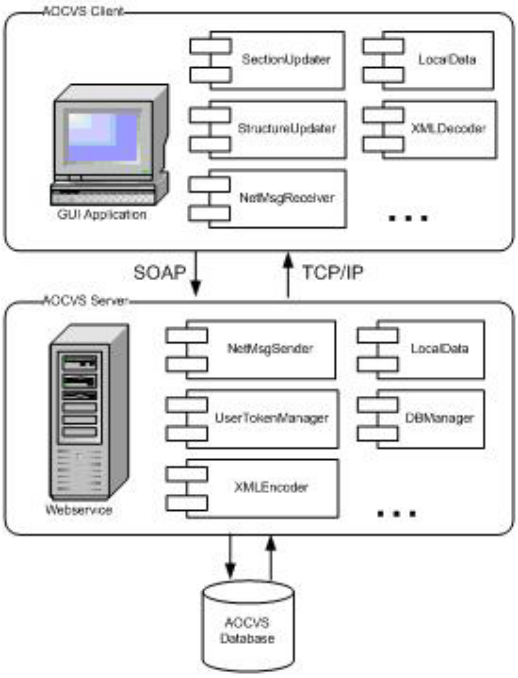

Figure 3 - Simplified architecture of AOCVS

5.2 Database based CVS

Unlike traditional CVS, AOCVS stores everything within the database; there are no physical files stored on the server. The database currently used by the system is Microsoft SQL. The rationale behind the database design decision is to allow the system to search and retrieve information more efficiently in comparison to the traditional CVS and at the same time this allows the system to take advantage of the fail safe, backup and replication mechanism that are provided by the Microsoft SQL database.

The normal CVS recognises each file as an entity and keeps track of changes made to the file. By using the database the AOCVS is able to break up the content of files and store them as different sections, each section is tracked and any changes made to each section are recorded.

This use of database increased the effectiveness and efficiency of versioning control, with the help of SQL script language each section and its history is able to be located, identified, created, and retrieved with ease.

The rationale behind recognising sections of file as the lowest level entity is because it encourages and enhances the practise of eXtreme AOCE programming by allowing the users to focus their tasks on specific sections for each release of AO iteration.

Each section stored in the database is labelled with an identified aspect. These aspects increase the understandability of the code and they are also the fundamental support for the cross-cutting concern.

The uses of database on section storage enable AOCVS to address the cross-cutting concern with ease. Each section can be easily referenced into another file

or even the same file when stored under a different specified aspect.

5.3 eXteme AOCE coding standard

The eXtreme AOCE methodology promotes highly functional and better characterised and categorised code. In order to achieve this, the implementation has to follow a high quality coding standard. The AOCVS was constructed with well defined sections embedded into the system and tagged with appropriate aspects. Each component's interface within the AOCVS is well commented to explain the purpose and function of the component, an example of this is shown in figure 4. These comprehensive procedures enhance the understandability and reusability of the code and thus allow the system to be refactored with ease.

```
/// <summary>
/// This component defines an object that can receive
/// update msg and dispatch that message to perform the
/// desired tasks.
/// </summary>
public interface IMsgListener{

    /// <aspect> Transaction Processing </aspect>
    /// <summary>
    /// This method allows the component to receive input
    /// messages from other components. Upon receiving
    /// message the message should be decoded and
    /// update the correct fields.
    /// </summary>
    void TransationProcessing_ReceiveMsg(string msg,
            int input1, int input2, int input3);
}
```

Figure 4 - Sample code from MsgListener component

5.4 Dynamic network update

One of the unique features of AOCVS is its synchronous merge function, whereby each client is able to see the changes made to a section in real time fashion. In order to achieve this, a dynamic network update function was implemented into the server of AOCVS. The server of the AOCVS is a webservice and it records the IP address and socket number on each of the client as they connect to the server. Each of the clients acts as a TCP/IP server and constantly listens to network inputs. This allows the server to dynamically choose which client to update and actively sends update strings to the clients via TCP/IP connection.

Once the NetMsgReceiver Component receives a TCP/IP input from the server it will then decode the message and dispatch it to perform the appropriate action. The dispatching of the messages received by the client via TCP/IP is done by various Updater components. This setup will also allow future developers to add message dispatching functions of the client with ease.

With the use of dynamic update functions embedded in the AOCVS the ownership of each code section can be transferred with ease. The AOCVS server keeps record of all the opened sections and their current owner. When a change of ownership occurs the server will dynamically broadcast the change to the clients that has the section opened. This simulates our eXtreme AOCE coding environment and pairs can swap the control by using the keyboard.

Whenever a section is modified, the changes made will be broadcasted dynamically to all the interested clients. But this function initially resulted in frequent and excessive network data flow. Redesign and refactoring was done using eXtreme AOCE to minimise the data flow across the network. Components such as FileToken and UserToken Managers reduced long repetitive string information to 32 bits integer. These added features increased the efficiency of our current dynamic network updating function of the AOCVS.

5.5 XML encoding and decoding

Since the server of AOCVS is a webservice, XML encoder and decoders were created for passing complex structured information to and from the server. This has great advantages over the other methods for passing complex data such as passing formatted string, passing array or continuous web method calls.

The use of XML like the one shown in figure 5 enables the data to be human readable and also allows future developers to use other components to interact with the data passed across the network.

```
<Group159>
 <folder name="ChatUpdater" id="1">
  <file name="IChatUpdater.cs" id="1">
   <section name="INIT" id="2">
    <aspect="Unclassified"/>
    <text>//still need to be created</text>
   </section>
   <section name="IChatUpdater" id="4">
    <aspect="Unclassified"/>
    <text>//will be done soon</text>
   </section>
  </file>
</Group159>
```

Figure 5 - Sample XML used for complex data

6 Discussion

We have used and tested eXtreme AOCE for developing large and complex software systems, including developing the AOCVS system that we described in this paper. There are certain minor areas where the principles of XP and AOCE do not form a complete fit, and we will discuss how eXtreme AOCE cleverly resolves these issues. We found this novel new methodology to be extremely useful, effective and efficient, and possess many new features to support agile programming. The AOCVS tool itself is a new technology that is designed for supporting eXtreme AOCE. Here we will discuss the issues surrounding eXtreme AOCE and AOCVS, and also possible future research and development in this area.

6.1 eXtreme AOCE issues and solutions

One of the key features of XP is its flexibility and its ability to cope with changes made to the project scope, therefore it is taboo for an XP development team to plan and design to fine detail the structure of the system they are designing for [2, 9, 13]; AOCE on the other hand demands well designed interfaces for the components up front. These two principles appear to oppose each other [1, 3, 10, 15]. However, eXtreme AOCE resolves these conflicts by simply going to the root of the problem and addressing the logical and fundamental mind sets of software developers when implementing code. It is commonly acknowledged that, eXtreme Programming or not, when implementing a method, any responsible programmer must be aware of the purpose and function of the method within the software that is yet to be implemented. All XP programmers are required to have the wider knowledge as to what the main purpose of the whole system is, and this knowledge from XP is use to feed the eXtreme AOCE development process.

From the feed back of the developers who used AOCE to develop software, it was discovered that if the developers lack organisation and communication, i.e. multiple different components with similar functions may be produced. eXtreme AOCE solves this issue by encouraging and enhancing the communication between the developers with the practise of XP.

Another concern with eXtreme AOCE centres around refactoring. XP encourages refactoring, but during the refactoring process it may be necessary to change the interfaces for the components [6] and this could cause an adverse code change chain reaction. However, this will not be an issue if the refactoring was done in a backward compactable manner paying attention to the aspects involved. It is our general practise not to modify the signature of the existing methods in a interface of exposed APIs, i.e. rather than changing the method names, we create new methods with new names and better and more efficient exposed functionalities through eXtreme AOCE.

As eXtreme AOCE is a combination and extension of both XP and AOCE methodology, it naturally inherits some of the pros and cons from both methodologies. As this combination also uses the strength of one methodology to make up for the weakness in the other, eXtreme AOCEA is far more advanced and effective than using XP or AOCE alone.

6.2 Draw backs on AOCVS

As mentioned previously, AOCVS is a tool designed especially for software development under the eXtreme AOCE methodology. It can be used for normal XP or AOCE development but not very well for other methodologies besides these.

Since AOCVS targets software development, information on formatting and fonts are discarded by the system. This makes the use of AOCVS on non-software projects very tedious, because it was not designed to do this.

The use of AOCVS also differs greatly from the traditional CVS and this may require some initial relearning among the users overly familiar with CVS.

6.3 Future development for AOCVS

AOCVS is a newly developed tool, hence we have many good ideas for its future development. Here we discuss some of the ideas from our future development brainstorm.

6.3.1 Portable plug-in. In following the current style of usage of AOCVS, it would be better if the engine of AOCVS can be embedded into a software development tool such as Visual Studio .NET as a plug-in. This would allow the developers to use AOCVS purely for collaboration and use a software implementation tool for compilation and coding support.

The AOCVS itself is constructed with highly portable components; therefore, with the help of the web service based technology, AOCVS can be made into a universal portable plug-in for various software development tools and this would make it extremely handy and extend its use.

6.3.2 Enhancing communication. Voice communication and video steaming should also be built into the AOCVS engine. Communication is the key to collaboration and team work, therefore technologies such as these will further increase the efficiency of the eXtreme AOCE software developers.

6.3.3 Probing. One of the most interesting and promising idea from the future development brainstorm is the idea of "probing". The concept of this idea is to probe into the files on the client's machine and detect changes made to the files. This idea opposes the concept of AOCVS of having only one copy of the file, but this idea would free the developers from having to upload and download files and at the same time being able to store the files on their local machine.

7 Summary

Our eXtreme AOCE methodology extended XP with the support for cross-cutting concerns and enhanced understandability, reusability and quality of the implemented code. By using eXtreme AOCE to drive a software project, the development team will benefit with added business value by building up more and more reusable aspect-oriented components in their software library.

The tool that we designed and implemented to lubricate development using the eXtreme AOCE methodology is the AOCVS. Unlike traditional CVS, AOCVS targets specifically at software project done using the eXtreme AOCE methodology. It provides rich

features to search and filter based on aspects and components. AOCVS simulates the pair programming environment with the function to transfer the editing control on a section inside a file. Synchronous merge style is the primary way changes to a file are merged on AOCVS, this function is an alternative to the traditional asynchronous merge done in CVS and it removes the hassle for developers having to deal with versioning difference at merge time.

Our work with eXtreme AOCE shows that it improves the agility of the agile software development, and coupled with the support from AOCVS it enables developers to work with greater effectiveness and efficiency.

8 Acknowledgements

We would like to thank the anonymous reviewers for their helpful suggestions. We also wish to thank all members of the *eXtreme AOCE Group* for their comments.

9 References

[1] Grundy, J., Panas, T., Singh, S. and Stöckle, H., An Approach to Developing Web Services with Aspect-oriented Component Engineering, NCWS03, 2003

[2] Beck, K., eXtreme Programming eXplained – embrace change, published by Addison Wesley, 2000

[3] Singh, S., Grundy, J. and Hosking, J., Developing .NET WebService-based Applications with Aspect-Oriented Component Engineering, AWSA'04, Australia.

[4] CVS – Concurrent Versions System: The open standard for version control, 2003, https://www.cvshome.org/

[5] Allen, p., and Frost, S., Component Based Development for Enterprise Systems: Applying the Select Perspective, Addison-Wesley, 1998.

[6] Fowler, M., Refactoring: Improving the Design of Exisiting Code, published by Addison Wesley, 1999; ISBN 0201485672

[7] Qu, C. and Nejdl, W., Construction a Web-based Asynchronous and Synchronous Collaboration Environment Using WebDAV and Lotus Same Time, 2001\

[8] Grundy, J.C. and Hosking, J.G., In Engineering plug-in software components to support collaborative work, Software –Practice and Experience, 2002; vol. 32, pp. 983-1013.

[9] Beck, K. and Andres, C., eXtreme Programming eXplained – embrace change, second edition, published by Addison Wesley, 2004

[10] Grundy, J., Multi Perspective Specification, Design and Implementation of Software Components using Aspects, In International Journal of Software Engineering and Knowledge Engineering. Vol. 10, No. 6 (2000) 713-734, World Scientific Publishing Company.

[11] Li, S.F., Fraser, Q.S., and Hopper, A., Integrating Synchronous and Asynchronous Collaboration with Virtual Network Computing, Internet Computing, May/June 2000, 26-33

[12] Kiczales et al, Aspect-oriented Programming, In Proc. Of the 1997 European Conf. on Object-Oriented Programming, Finland (June 1997), Springer-Verlag, LNCS 124.

[13] Beck, K. and Fowler, M., Planning Extreme Programming, published by Addison Wesley, 2001

[14] Cederqvist et al, Version Management with CVS, published by Network Theory Limited, 1992

[15] Grundy, J.C., Supporting aspect-oriented component-based systems engineering, In Proceedings of 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, June 16-19 1999, KSI Press, pp. 388-395.

[16] Berliner, B., CVSII: Parallelizing Software Development, 1990, USENIX 1990 Technical Conference

[17] Frakes, W. and Fox, C., "Sixteen questions about software reuse", Communications of the ACM, 1995, 38(6):75-87, 1995.

[18] Miles, V.C., McCarthy, J.C., Dix, A.J., Harrison, M.D. and Monk, A.F., Reviewing Designs for a Synchronous-Asynchronous Group Editing Environment, 1993, In Computer Supported Collaborative Writing Ed. M. Sharpless. Springer-Verlag. Pp 137-160

[19] Dix, A.J. and Miles, V.C., Version Control for Asynchrous Group Work, 1992, YCS 181, Department of Computer Science, University of York, YO1 5DD

[20] Collaborative Editing for Text Editor, 2004, http://docsynch.sourceforge.net/

[21] Garmus, D. and Herron, D., Function Point Analysis: Measurement Practices for Successful Software Projects, published by Addison Wesley, 2000

[22] Abrahamsson, P., Extreme Programming: First Results from a Controlled Case Study, 1993, 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)