What are the benefits of Aspect Oriented Programming to project iterations developed using Agile Processes?

Joseph Muller

ABSTRACT
Aspect Oriented Programming (AOP) is a software development programming technique that facilitates the modularization of cross cutting concerns, for example logging, security and error handling [4]. Agile Processes, such as eXtreme Programming and SCRUM are software development methodologies that promote iterative and incremental design and development to projects [2].

This paper will test the hypothesis that no reduction in coding effort is observed when using AOP in Agile Process iterations as opposed to using standard coding mechanisms, providing quantifiable information that can be used and extrapolated by projects to determine the impact (reduction / increase) in time as a result of using AOP.

Keywords
Aspect Oriented Programming, AOP, AspectJ, Agile

1. INTRODUCTION
In recent years the demands on the development of software have increased greatly. Software development is now required to produce increasingly complex software, in shorter periods of time, with greater quality and fewer resources. As a result of these pressures, software development tools and methodologies have continued to evolve to address these issues.

Although AOP has been around for some time, the software development community is just beginning to evaluate and use AOP on software development projects only within the last couple of years. As software has become more complex, the amount of code required has increased. Software today can contain hundreds or thousands of modules and classes. Simply adding certain functionality such as logging for entry and exit into a module can result in a large amount of code being created. This type of change could possibly result in all modules or classes being affected. Since AOP modularizes what is called cross cutting concerns, that is functionality required across multiple sections of code, it has the ability to simplify the code required, as well as decrease the amount of development time required. Code that would normally be added to every class or module can be created once and utilized across all modules or subset of modules.

Another change in the development of software has been the introduction of new development methodologies classified as Agile Processes. Examples of Agile Processes include eXtreme Programming, Feature Driven Design (FDD) and SCRUM. The whole premise behind Agile Processes is that software development should be iterative and incremental [2]. In addition software is developed based on requirements collected during the iteration being executed and not based on any future requirements. This allows the software and the development process to adapt to new features, functionality and requirements, however results in issues when these features or functionality need to cross many modules or classes. In cases where a feature or functionality is required across many modules, work is required in each module or class, resulting in additional time expended to design, code and test each module affected.

In looking at software development lifecycle, many tools, technologies and methodologies can be used to address the pressures of complex software, quicker time to market and improved quality. Projects developed using Agile Process methodologies as well as Aspect Oriented Programming, will be able to gain the benefits provided by both. An Agile Process will provide for iterative and incremental development, resulting in a quicker time to market. In addition since software is developed in collaboration with the customers and iterations are delivered frequently, the expectation is that the functionality meets the customer’s needs or can easily be adapted to meet their needs, resulting in increased quality of the final product. The use of AOP helps to address one of the issues with Agile Processes, that is features or functionality that are system wide, or affect all or most of the modules in a system. Non-functional requirements introduced into an Agile Process project, will normally have a large impact on the project. Since Agile Processes do not take into account future or unknown requirements, the possibility exists that changes would be required throughout the system if not for AOP. AOP allows these changes to be modularized into a central location, limiting or possibly eliminating the amount of change that would be normally required within the affected modules.

1.1 AOP
Aspect Oriented Programming (AOP) was originally developed in the late 1980’s at the Xerox Palo Alto Research Center (PARC) [5]. All software that is developed has concerns. A concern is a goal, concept or area of interest. In most software there are two types of concerns, core concerns and system level concerns [8]. Previous development methodologies such as Object Oriented Programming (OOP) have done a good job of encapsulating core concerns into modules or classes. The issue with Object Oriented Programming however occurs when concerns need to span more that one module or class. These system level concerns are also known as cross cutting concerns. Common examples of cross cutting concerns are logging, security and tracing. Using Object Oriented Programming these concerns would be coded and added to each class and module where required, making the resulting code harder to read, difficult to maintain, understand and reuse. AOP however handles these concerns without having to modify each class or module affected. AOP complements Object Oriented Programming by facilitating modularity and pulling the implementation for cross cutting concerns into a single unit [4].

Although the theory of Aspect Oriented Programming is good, without an implementation available for projects to use, it would not be very helpful. In addition to the development of AOP, Xerox PARC created AspectJ as an extension to the Java language to handle the implementation of AOP [4]. AspectJ is just...
one implementation of AOP available in the market today. A number of AOP implementations are now available, not only for Java but other languages as well. Some examples include AspectJ for .NET, Aspect C++ for C++, and AspectWerkz for Java. A listing of available tools can be found on the Aspect Oriented Software Development site [9].

Regardless of the implementation, there are certain concepts that are standard.

An implementation of AOP will require the following:

- **Joinpoints** - well-defined points within a program, for example method calls, method entry, method exit and exception handling blocks [4].
- **Pointcuts** - collections of one or more joinpoints that will be acted upon [4].
- **Advice or Introductions** – Advice is the code that is executed based on a set of specific joinpoints and a specific context. Introductions provide the ability to affect the static code of a program with the introduction of additional attributes or methods [4].
- **Weaver** – responsible for taking the information specified for joinpoints, pointcuts, advice / introductions and the system code, to create a functional system by incorporating the AOP code into the standard code.

It is important to note that not all implementations support functionality defined within AOP. For example AspectJ does not allow joinpoints to be specified for conditional checks and looping constructs [7]. Even with these limitations, the implementation of AOP still provides benefits over standard OOP implementations. These benefits are the modularized implementation of cross-cutting concerns, easier evolution of systems, late binding of design decisions and increased code reuse [6].

Coding that will be done as part of the research to determine the benefits of using AOP in an Agile Process project iteration will use Eclipse 3.0 in combination with AspectJ [1] as the implementation of AOP.

### 1.2 Agile Processes

Over the years many software development methodologies have been created. One of the early methodologies is commonly referred to as the waterfall approach. It is named as such because the overall premise is that one stage only starts after the prior stage has completed. So design would only start after all requirements have been completely detailed out and coding only starts after the design for everything had been finished. In many cases this results in a software system being delivered many months or years later. In addition since the system is not delivered until the end there is a risk that the final product will not meet the customer’s needs or requirements. Another popular methodology that has been adopted and used is the Rational Unified Process (RUP). The intention of RUP was a methodology that promoted iterative development. That is requirements; design, coding and testing would be done in smaller iterations, allowing quicker delivery to customers, eliminating some of the risks involved with the waterfall approach. While RUP solved issues encountered as a result of the waterfall approach, RUP is considered a framework that allows projects to use and customize the methodology as they see fit. As a result many projects became so concerned over producing the documentation specified and required as part of RUP, that the delivery of code would actually become secondary. In these cases focus was not on the end product, the system for the customer.

As a result of continuing problems in software development, and the need for alternative methodologies to the existing document centric, heavy weight methodologies, a number of industry leaders in software development gathered in 2001 and founded the Agile Alliance [10]. Initial members of the Agile Alliance included Kent Beck, Martin Fowler and Alistair Coburn to name a few. As a result of the meeting, the Agile ‘Software Development’ Manifesto was born [3]. The bases of the Agile Manifesto are the following values:

- **Individuals and interactions over processes and tools**
- **Working software over documentation**
- **Customer collaboration over contract negotiations**
- **Ability to respond to change**

Out of the core values defined, a set of principles evolved. These principles are welcoming changes to requirements; frequent delivery of software; design and build for know requirements and a close partnership between customers and developers. There are a number of Agile Processes available to projects and developers today, including eXtreme Programming (XP), SCRUM and Feature Driven Development (FDD). The main theme with all of these is that development should be incremental and adaptable, providing working versions of software in rapid iterations.

By working closely with the customer, designing and coding for existing requirements and providing delivery of software often in smaller iterations, Agile Processes look to avoid the risks that are inherent in other methodologies used over the years. The downside that has been identified with these Agile Processes however is the effect of non-functional requirements, especially when they are introduced in later iterations after parts of the system have already been developed. These non-functional requirements, examples being logging, auditing and error handling, often affect large sections of the existing modules developed. And although Agile Processes welcome requirements changes and are made to be adaptable, requirements or functionality that requires changes to large numbers of existing modules still can require a large amount of effort.

### 2. AOP in Agile Process Project

#### 2.1 Process

Based on what we know regarding AOP and Agile Processes, there are complementary benefits for using both within the same project.

The question becomes, what is the quantifiable benefit that can be cited and calculated by project teams looking to use both an Agile Process and AOP together on a single project. In order to effectively measure the affect AOP has on an Agile Process project, we will need to calculate the impact of AOP on design
and coding time, total lines of code required, testing time and maintainability of the code. These should all be measured in comparison to the use of standard OOP coding practices.

The first step in calculating the impact of AOP and comparing it to standard OOP coding practices is to start with a set of five standard services that would commonly be found within a business application. These five services are:

- Get Customer – provided with a unique customer identifier the service would return customer information currently available including name, address, phone number and email address.
- Create Customer – provided with required customer information to include unique identifier, name, address, phone number and email address, a new customer record would be created and stored.
- Maintain Customer – provided with a unique customer identifier as well as other customer information (i.e. name, address, phone number and email address), the record would be updated in the database with the new information.
- Get Customer Accounts – provided with a unique customer identifier, retrieve a listing of customer accounts including the account identifier and account description.
- Get Customer Balances – provided with a unique customer identifier, retrieve a listing of customer accounts by account identifier and the associated balance for the account.

The five services will be developed as stateless Java services and will be the basis for the changes required in order to quantify the difference in using AOP. All development that is required will be accomplished using Eclipse 3.0.

Once these basic services are available, each service will go through four additional iterations of design, coding and testing. This is similar to work that would occur in an Agile Process project where iterations would add functionality and features. The four additional iterations will add the following:

- Authentication – will ensure that each service called has the appropriate security token prior to execution of the service functionality.
- Auditing – will capture each execution of a service, including details around the parameters supplied.
- Logging – provides the ability to log the start and end of each service.
- Error Handling – provides a standardized mechanism for the handling of errors within each service.

Each of the four additional iterations will be coded in two ways.

1. Standard Programming – Changes for the iterations will be designed and developed in line with standard programming, more specifically Object Oriented Programming and Service Based Programming. Coding will be handled by the Eclipse 3.0 IDE.
2. Aspect Oriented Programming – Changes for the iterations will be designed and developed using Aspect Oriented Programming. Coding will be handled by the Eclipse 3.0 IDE in combination with the AspectJ Plug-In to handle the AOP extensions to Java.

As each iteration is being developed under the two different methods, the following information will be captured so that quantitative information will be available.

1. Time to Design and Code Changes – The time to design and code changes will be tracked in minutes.
2. Number of lines of Code – The number of lines of code within each method will need to be counted and compared to the total lines of code. The definition of lines of code includes the following:
   - Physical Source Lines of Code (SLOC), which would be non-blank, non-comment lines. Code Counter Pro by Gerone Soft will be used to count source lines of code.
   - Configuration / XML Files containing non-blank, non-comment lines will also be counted. Based on the implementation it is possible that configuration files would be used to control processing and therefore need to be included in total as part of the solution.
3. Testing time required – The Amount of time required for testing of changes will be tracked in minutes.
4. Maintainability Ranking – A ranking of the maintainability of the code will be assigned. These rankings will be selected from the following categories (Very Complex, Complex, Moderate, Easy and Very Easy).

The assigning of a maintainability ranking will be a subjective measurement, unlike the other three which can be measured objectively. The following definitions will be used in regards to the maintainability ranking categories.

- Very Complex – Indicates the code for the software needs to be maintained by an experienced, most senior level developer, familiar with the existing design and source and that any changes require additional effort.
- Complex – Indicates the code for the software needs to be maintained by an experienced, senior level developer, familiar with the existing design and source.
- Moderate – Indicates that the code for the software can be maintained by an experienced developer, not necessarily the most senior developer and any changes should require some amount of effort, although not a large amount.
- Easy – Indicates that the code for the software can be maintained by an entry level developer with a little guidance. Any changes require a minimal amount of effort.
- Very Easy – Indicates that the code for the software is simple to understand and modify. Code could be maintained or modified by an entry level developer.

Once these iterations are completed, the data collected will be used in determining the quantifiable benefits of using AOP versus standard coding mechanisms within Agile Process iterations.
Although the sample size is limited by costs and time, results from these sample services and sample iterations can be extrapolated to specific projects based on the number of services or modules contained within the project.

2.2 Results
An experienced Java developer, familiar with not only the Eclipse 3.0 development environment, Java, Aspect Oriented Programming as well as a number of different development methodologies, developed the five services previously cited (GetCustomer, CreateCustomer, MaintainCustomer, GetCustomerAccounts and GetCustomerBalances). Once these base services had been created, the developer did the design, coding and testing for additional iterations. These iterations were accomplished using a standard methodology as well as AOP. The iterations involved the following:

- Iteration 1 – Authentication
- Iteration 2 – Audit
- Iteration 3 – Logging
- Iteration 4 – Error Handling

The following table shows the results for the design and coding, in number of minutes.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design – Standard Methodology</td>
<td></td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>45</td>
</tr>
<tr>
<td>Coding – Standard Methodology</td>
<td></td>
<td>20</td>
<td>10</td>
<td>12</td>
<td>20</td>
<td>62</td>
</tr>
<tr>
<td>Design – AOP</td>
<td></td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>29</td>
</tr>
<tr>
<td>Coding – AOP</td>
<td></td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td>47</td>
</tr>
</tbody>
</table>

Based on the information collected, we can see that the total design and coding time using a standard methodology was a total of 107 minutes, where as the AOP Methodology was 71 minutes.

By looking at the amount of time in design, coding and testing, the AOP Methodology saved 49 minutes of time over the standard methodology.

The last objective piece of information to be collected was the Source Lines of Code (SLOC) between the two methodologies. In order to determine this, Code Counter Pro from Gerone Soft was used to count the source lines of code in the original base code, as well as the standard methodology and AOP Methodology coding after the four iterations were completed. These results are summarized below.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>361</td>
</tr>
<tr>
<td>Standard</td>
<td>462</td>
</tr>
<tr>
<td>AOP</td>
<td>434</td>
</tr>
</tbody>
</table>

After the iterations had been completed for both the standard and the AOP methodologies, the maintainability ranking of each methodology was collected by asking the developer. This ranking was selected from the previously defined categories of Very Complex, Complex, Moderate, Easy and Very Easy. While this is not an objective measure like the time to design, code and test; or the source lines of code, it is valuable to help in understanding the complexity of the final code and the skill set required to code and maintain.

The following rankings were assigned by the developer based on the developer’s skill, experience and work on the iterations with standard programming and AOP. The standard programming was ranked as Easy, since all the code required is implemented within each method or service and can easily be maintained by an entry level programmer with minimal guidance. The AOP development was ranked as Moderate, due to the complexity involved with the aspects that need to be created. Although the coding for the aspects in itself was not complex, an understanding of the join point patterns as well as understanding and realizing that not all the code to be executed resides within a module and in most cases is not even referenced within the module affected.
In summarizing the results we can see that the iterations developed using AOP resulted in reduced design, coding and testing times over iterations using standard programming. Overall using AOP resulted in these processes taking 49 less minutes using AOP, due to the reduced amount of code required, as well as the reduction in the number of modules impacted by the changed. This reduced amount of code was also evidenced by the Source Lines of Code (SLOC) that was counted for each of the methodologies. The use of AOP resulted in 28 less lines of code being developed over standard programming methodologies. While these results do not seem to be very large, it is important to keep in mind that this work was done only using five very basic services and four iterations for basic system level requirements. If these results were used to calculate the impact on a system that required hundreds or thousands of services, the savings in time for development could easily result in the savings being in the hundreds of hours, resulting in software being delivered quicker with a reduced cost.

3. Conclusion
Aspect Oriented Programming and Agile Process iterations each provide unique and complementary benefits in the development of software and should be used together, over Agile Processes with standard coding practices. Agile processes provide for iterations and incremental development, resulting in the finished product being delivered incrementally and in many cases getting a working product quicker. AOP provides the encapsulation of logic that would otherwise be spread across many modules, resulting in reduced lines of code created, and consequently reduced development times. This was shown with a reduction in code created and time required to design, code and test iterations.

Although Agile Process iterations can and are developed without using AOP, the benefits from AOP, especially in regards to system level requirements, required across many modules in the system, can be significant. The research and coding conducted shows that with five simple services and simple iterations to add basic system level concerns, there was a 25% reduction in source code created. Projects can use the reduction in source code as well as the savings in time required to design, code and test these system level requirements to determine the full impact to their specific project.

Projects looking to use Agile Processes and AOP, should also take into account some additional information not specifically covered within the scope of this paper when making a decision. Although the standard concepts around AOP are the same, each implementation of AOP is different. Code that is developed within AspectJ for handling aspects is different than code developed within JBoss or Aspectwerkz. If the organization has multiple development environments, or is planning on changing development environments in the near future, the non-standard implementation of AOP can be an issue. In addition development using AOP means that code that was previously scattered across many modules is not centralized. Developers making changes to the code need to be aware of the impact these changes can have. Simply changing the pattern or join points can have an enormous impact on the system, possibly resulting in code no longer being executed when it should be. Rigorous testing needs to be followed to ensure that changes to the points or advice have the appropriate affect. The last point that needs to be kept in mind with the use of AOP is how much of the functionality available with AOP is actually used. One of the features available within AOP is the ability to add attributes and methods to existing classes. This feature is allowing the dynamic modification to the static code. While there may be situations where this is valid, use of this type of feature needs to be weighed carefully against the impact to the testing and future maintenance of this type of coding.

AOP used within Agile Process iterations can be beneficial by reducing the amount of code required, as well as time required to design, code and test. Projects should look to use both, keeping in mind issues and concerns raised around rigorous testing due to the large reaching impact of changing aspects as well as guidelines around when and where AOP should be used within the project.

4. REFERENCES