World of Computer Science and Information Technology Journal (WCSIT)
ISSN: 2221-0741
Vol. 3, No. 3, 150-155, 2013

Aspect Oriented Programming in Early Stages of a Software Development Project

Luis Marco Cáceres Alvarez, Marjane Delgado
Escuela Universitaria de Ingeniería Industrial, Informática y de Sistemas, University of Tarapacá
Arica, Chile

Mauricio Arriagada-Benítez
Department of Computer Science, School of Engineering,
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract—Nowadays, the aspect-oriented programming is increasingly taking more importance and many companies are opting to use in software development by all the benefits it offers. However in most cases this is used only in the construction phase without being considered in the previous stages to it, making difficult to have an appropriate documentation with development and not being considered from the beginning where aspects are disorderly placed. This work proposes the possibility of using aspect-oriented programming from early stages of the software's life cycle, like the managing of requirements. For this demonstration was applied the paradigm of aspect-oriented programming in an application model, using RUP to represent the inclusion of these elements in the different stages of project life cycle of software. After performing a statistical sampling in a company that makes formal software development based on the set of McCall´s Software metrics concerning the revision and product transition, we obtained favorable results in various aspects related to the proposed model. Our findings show that the use of the POA can be used from early stages of a software project and that could obtain considerable improvements in maintenance, flexibility, ease of test portability, reusability and interoperability of the software.

Keywords- Aspect-Oriented Programming; RUP; separation of crosscutting concerns; McCall Metrics; Software Engineering; UML extensions.

I. INTRODUCTION

Current methodologies are mostly designed to work with OOP (Object Oriented Programming), which has become trendy. However, within the last few years, a new programming paradigm has arisen, and that is AOP (Aspect Oriented Programming).

AOP complements OOP, allowing the developer to dynamically modify an OOP model, to create a system that can grow to fulfill new requirements. The same as in real life, objects can change their status during their life cycle; an application can adopt new features as it evolves [1].

All elements spread in the system, or aspects, overload some objects, separating them from their main function. With inappropriate functions, these aspects are increasingly being used to "clean" objects, since they remove all crossed functions among these, and are handled within these aspects.

The aspects could be even considered from the process of requirement choice on, despite this, there is a scarcity of methodologies that incorporate AOP through their life cycle [1][14].

As it is known, one of the most used software development cycles is Rational Unified Process (RUP) [2] because of its Unified Modeling Language use, and also due to its adaptability to different companies in the industry.

Following this further, the proposition is to apply the AOP in an application, using RUP methodology as the basis, to represent the inclusion of these elements in different stages of its life cycle in a software project.

II. THEORETICAL FRAMEWORK

One of the first concepts which appeared defining what an aspect is, was stated by Lopez and Lieberherr [3] in which is explained that "an aspect is a unit that can be defined in terms of partial information of other units". However, the definition of aspect has evolved in time. Currently, there is an agreement about the definition of aspects that is given by [4].

"An aspect is a modular unit that is spread along the structure of other functional units. Aspects are found from the designing stage up to the implementation stage. A design aspect is a modular design unit that is mixed in the structure of other parts of the design. A program or code aspect is a

modular unit of the program which is present in other modular units of the program".

In this same manner, according to [5] in a more informal way provides the following definition:

"Aspects are the basic unit of AOP, and they can be explained as parts of an application which describe the key questions related to the essential semantics or performance. Also they can be seen as elements spread all over the code, which are difficult to describe locally compared to other components"

In figure 1 [5] we can observe the structure of a basic aspect oriented program, where the model of objects is shown apart from the different aspects, as: memory management, synchronization, error management, distribution, etc.

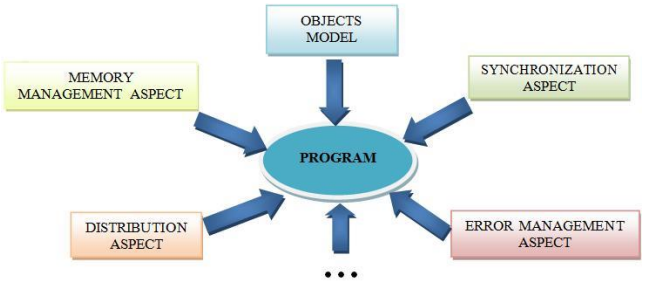

Figure 1. Aspect Oriented Program structure.

This methodology is present in different stages of software development as [6] explains, which when revised by [7] shows conflicts and bad practices that can occur in this extent.

Software design is seen by Apel [8][13] as a stage in which software design can be improved. Afterwards, Przybylek [9] presents UML extensions to AOP, service oriented programming, among others.

This paradigm has been used in companies [10] but only under a development perspective, and not in early development stages.

Within this context, the AOP paradigm is presented in the following sections, in early stages of the software engineering development cycle, as it is required.

III. Analysis and Design

To be able to include aspects in the process, the project to be used will be analyzed, designed, and built according to the RUP methodology. Finally it will be evaluated to determine the obtained benefit.

The project to exemplify is a Bank Rate Administrator, which consists on a Data Entry to assign the rates to each available operation, allowing editing previous rates, and the categories which those belong to.

A. *Requirement taking*

Requirement taking is presented through this use case diagram (figure 2) in which the eight different actions available from this application are presented.

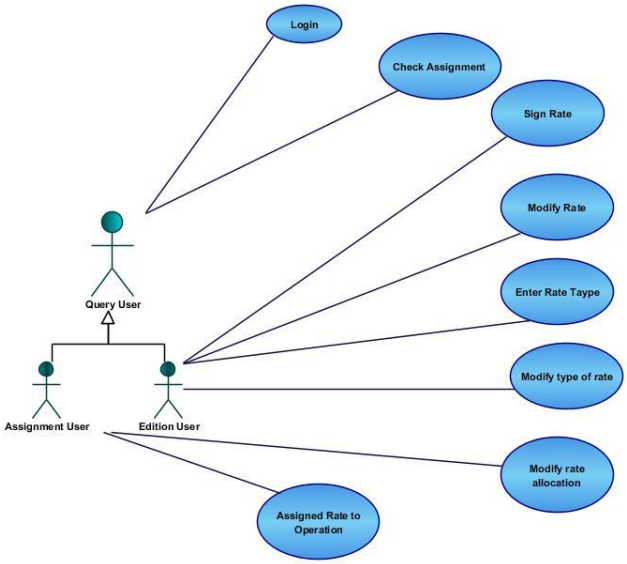

Figure 2. Use case diagram.

In figure 2, the use case diagram, we can see the different functions this application will do. The importance of this diagram is in its specification, where from the point of requirement taking, we can identify the aspects as non-functional requirements, especially if these are repeated in different use case.

Two transversal functions or non-functional requirements can be identified in the diagram: the need of a log or application record, and tradability, for future operations.

B. *Class diagram*

For the class diagram, Pryzbytek's work [9] was taken as an example. The current UML extension mechanisms were chosen, the lightweight mechanism for the representation of aspects, since this is simpler and enough for the analysis representation.

According to [9] a lightweight mechanism is the one which does not need to define new elements in the UML existing metamodel. To be able to use those existing ones, you have to define profiles.

Likewise, an UML profile is a pre-defined stereotypes group, labeled values, restrictions and graphic icons which allow a specific domain to be modeled. [11] It was defined to provide a lightweight extension mechanism. The intention of the profiles is to give a direct mechanism to the adaptation of the UML metamodel norm, with constructions that are specific of a determined domain. [9]

The following figure 3 represents the class diagram additionally to the layers as the available aspects.

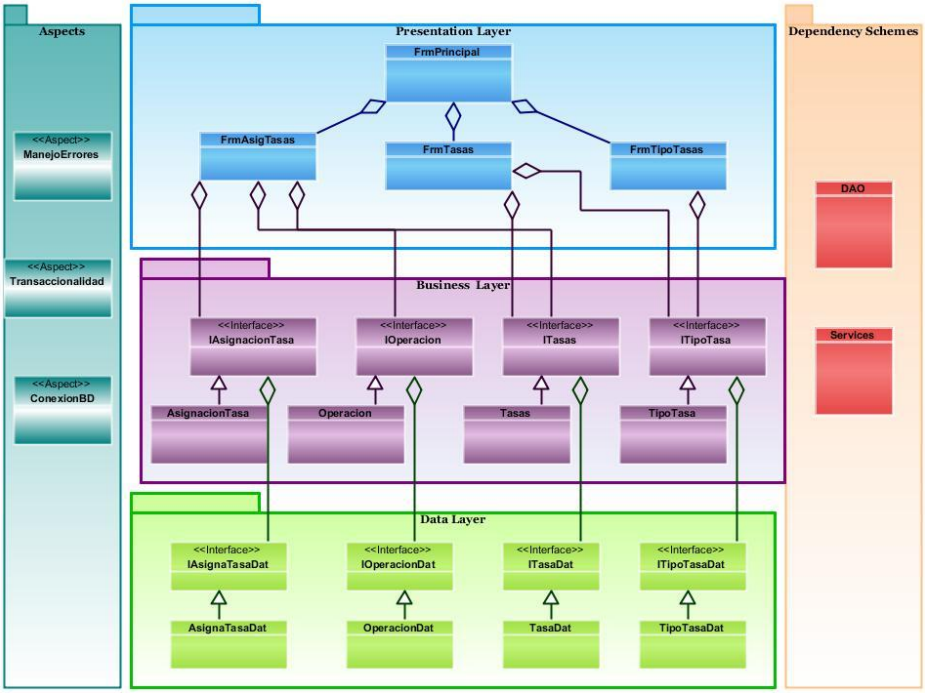

Figure 3. Class diagram.

The class diagram allowed a broader vision of the structure, as it was presented in layers; it allowed identifying the function and structure of each of them. In the same way, the aspects are present as well, although because of being a static model, the interaction is not yet observable.

C. *Sequence diagram*

The following figure 4 shows the use case sequence diagram of rate entry, to be able to appreciate the interaction among the classes and the aspects in a dynamic way.

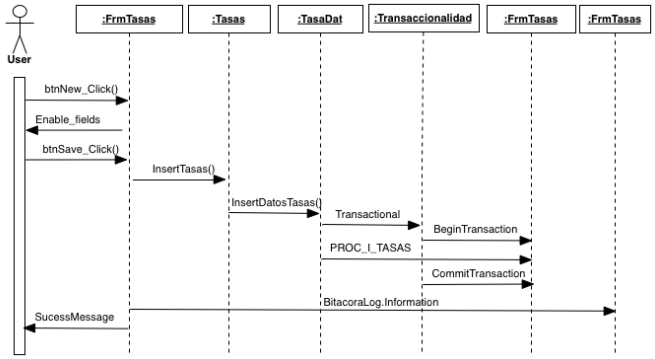

Figure 4. Sequence diagram.

Rate (Tasa) entry begins in the presentation layer, in the FrmTasas form with the user clicking on the new button. Immediately, the space to fill with rate entry data will be habilitated, and savable by one click. With this step, the entry application is sent to the business logic layer, and then to the data logic layer.

In the data layer, the function cited is marked with the tradability aspect, for which before executing the function, an invocation to the aspect is made, to initiate the transaction in the database.

As the aspect controls the process, is not required to have a return value. In case it fails, the aspect would take care of the rollback, and the error manager would send the error to the main window.

Finally, the FrmTasas, storages in its record the successful transaction and shows a satisfactory message to the user.

IV. IMPLEMENTING

Implementation of this research is as follows:

A. *Aspects*

The application was made considering three perspectives: tradability, "Bitacora" and BD connection. To represent the aspects, we used the C Sharp.Net language, along with the FrameworkSpring.Net.

For all the aspects, the XML scheme was used, because of its flexibility and easy setting.

1) Data Base aspects: The following (figure 5) is an XML scheme of the data base aspect, it mainly defines:

```
<?xmlversion="1.0"encoding="utf-8" ?>
<objectsxmlns="http://www.springframework.net">
<object
    id="PROV_FRMK"type="BCP.FRMK.COM.AA.Stub.ConfBD
    .StubDbProvider, AutenticacionAutorizacion">

<propertyname="TargetDbProvider"
ref="BD_TASAS"/>

<propertyname="CadenaDeConexion"
value="
    Data Source=MARJANELAP\SQL2005;Initial
    Catalog=TasasBD;
    Integrated Security=SSPI"/>

</object>
</objects>
```

Figure 5. Aspect XML scheme, database connection.

- Object Id: is the name it will be given to the aspect, in this case we use the name "PROV_FRMK".
- Object type: the name space is placed where the aspect implementation is located, followed by AutenticationAutorización which is the name of the project in charge of the aspect management.
- Properties: as properties it requests the name of the provider for the database to which is connected, in this case "BD_TASAS", followed by the connection chain used to reach to the database.

2) Bitacora aspect: The Following, figure 6, is an XML scheme of the Bitacora aspect, in this one are mainly defined:

```
<?xmlversion="1.0"?>
<objectsxmlns="http://www.springframework.net"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:aop="http://www.springframework.net/aop">

<aop:config>

<aop:advisor
    pointcut-ref="mensajesDeErrorAttributePointcut"
    advice-ref="mensajesDeErrorAdvice"/>
</aop:config>

<object
    id="mensajesDeErrorAttributePointcut"type="Spri
    ng.Aop.Support.AttributeMatchMethodPointcut,
    Spring.Aop">

<property
    name="attribute"value="BCP.FRMK.COM.BT.Bitacora
    ErrorHandler, Bitacora" />
</object>

<object
    id="mensajesDeErrorAdvice"type="BCP.FRMK.COM.BT
    .BitacoraErrorHandler, Bitacora"
    singleton="false"/>
</objects>
```

Figure 6. XML scheme of the Bitacora aspect.

- Aop Advisor: main indication of what an aspect is about, it will be addressed as "mensajesDeErrorAtribytePointcut (errormessageAtribytePointcut) for the pointcut and "mensajesDeErrorAdvice" (errormessageadvice) for the advisor.
- Pointcut object Id: the refered name is repited as pointcut "errormessagetribytePointcut".
- Object type: refers to the manager, followed by the library that supports it, it is AtributeMatchMethodPointcut, followed by Spring.Aop.
- Properties: as a property, there is the object to be used, it is about the class and project that supports it, in this case is "Bitacora", with its name BCP.FRMK.COM.BT.BitacoraErrorHandler". In the same manner, for the advisor definition.
- Advisor Id: the referenced name is repeated as advisor "erroradvicemessage".
- Advisor object type: it is referenced to the manager to be used. It has to be the same used as the Pointcut property.

3) Tradability aspect: In figure 7, we find the XML scheme of the aspect, in which are mainly defined:

```
<?xmlversion="1.0"encoding="utf-8" ?>
<objectsxmlns="http://www.springframework.net">

<object
    id="transactionManager"type="Spring.Data.Core.A
    doPlataformTransactionManager,Spring.Data">

<propertyname="DbProvider"
ref="PROV_FRMK"/>

</object>
</objects>
```

Figure 7. XML scheme of the tradability aspect.

- Object Id: it is the name given to the aspect, in this case we are using the "transactionManager" name.
- Objet type: the manager name is used where is pre-defined, followed by the library which supports it, as it is seen in AdoPlataformTransactionManager, followed by Spring.Data.
- Properties: in this case, another aspect will be used as a provider, PROX_FRMK which allows connection to the database in which the transactions will be opened and done.

V. EVALUATION METHODOLOGY

To assess this AOP proposal, a statistic sample was done, which allowed measuring the obtained results. In this manner, we used software quality measures and the most relevant quality factors, described by McCall [12] as it is on Figure 8.

Questions to be Assessed \ Quality Metrics	Maintenance	Flexibility	Testability	Portability	Reusability	Interoperability
Auditability			*			
Communication standardization						*
Completeness	*	*				
Concision		*	*			
Consistence	*	*				
Data standardization		*	*			*
Expansion capacity		*				
Generality		*			*	*
Hardware Independence				*	*	
Instrumentation	*	*	*			
Modularity		*	*	*	*	*
Simplicity	*	*	*		*	*
Systems Independence			*	*	*	
Traceability				*	*	

Figure 8. McCall metrics.

According to the INEI1 the staff population who develops software is about 7800 people in Peru, divided in 11 different functional areas. This collects a sample on 2 of these areas: management/direction and analysts/programmers, corresponding to figures of 858 (11%) and 2184 (28%) of the population.

The number of surveys applied is estimated by using equation 1 for stratified sample.

$$n = \frac{\left(N * Z_{¥}^2 * p * q\right)}{\left(e^2 * \left(N - 1\right) + Z_{¥}^2 * p * q\right)}. \quad (1)$$

The parameters used for the selection of numbers of sample by programmer are:

- N = total population number 2184
- Z_{∞}^2 = 1.65 (for a 90% accuracy)
- p = satisfactory expected proportion (in this case = 0.98)
- q = 1 – p (0.02)
- e = 5%.

Through a pre filtered, we selected the sub population to make the final survey. The survey instrument questions are oriented to revision and product transition (the areas to be assessed) using McCall assessment scale [12] in a 1 to 10 scale, where 1 is the minimum and 10 is the maximum.

In figure 9, you can see the correspondence among the assessed factor codes and the questions in the survey.

[1]INEI: National Institute of Statistics and Informatics. Perú. URL: http://www.inei.gob.pe/

Questions	Factor to be assessed
A	Auditability
B	Communication standardization
C	Completeness
D	Concision
E	Consistence
F	Data standardization
G	Expansion capacity
H	Generality
I	Hardware independence
J	Instrumentation
K	Modularity
L	Simplicity
M	System independency
N	Traceability

f9. Factors to be assessed.

Question	Factor to be assessed
U	Maintenance
V	Flexibility
W	Testability
X	Portability
Y	Reusability
Z	Interoperability

Figure 10. Metrics to be assessed.

By the codes for questions and equivalences, we can obtain the metrical percentages through the following formula in figure 11.

Metrics to be assessed	Questions
Maintenance	$= \frac{C + E + J + L + U}{5}$
Flexibility	$= \frac{C + D + E + F + G + H + J + K + L + V}{10}$
Testability	$= \frac{A + D + F + J + K + L + M + W}{8}$
Portability	$= \frac{I + K + M + N + X}{5}$
Reusability	$= \frac{H + I + K + L + M + N + Y}{7}$
Interoperability	$= \frac{B + F + H + K + L + Z}{6}$

Figure 11. Metrical equivalences.

VI. Result Evaluation

After applying the survey, the questionnaires were classified according to their product: revision (maintenance, flexibility and testability) and transition (portability, reusability, interoperability) as show in Table 1.

TABLE I. Obtained results

Classification	Metric	Score
Product revision	Maintenance	8.7
	Flexibility	8.93
	Testability	8.98
Product transition	Portability	8.95
	Reusability	9.25
	Interoperability	9

Figure 12 shows that the Testability has the highest score in the product revision classification item, while the reusability category has the highest score in the product transition item.

The high scores evidence the importance of these metrics to improve software development using AOP in early stage of a software development project.

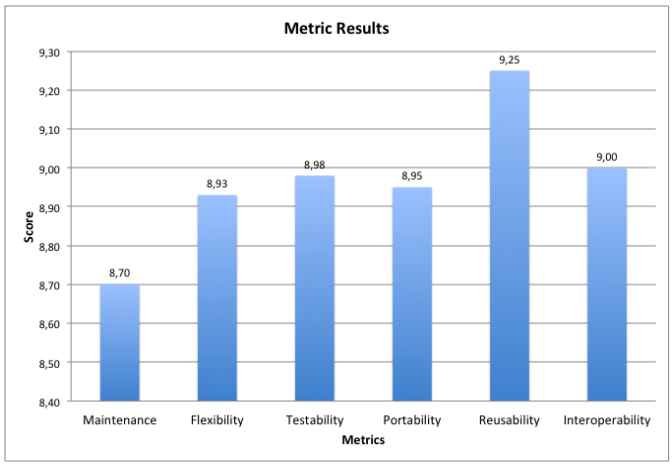

Figure 12. Graphic of evaluated metrics percentage.

VII. CONCLUSIONS

Considering the development of the TASAS application, we can state that it is possible to include the use of aspects from early stages of the process as the requirement analysis.

The relevance of this research is to demonstrate that this implementation brings concrete benefits to the following aspects: maintenance flexibility, testability, reusability, portability and application interoperability.

One of the suggestions to this research is that, considering that it was done with an application in C Sharp.net, for which the research should be repeated in other languages of programming so as to verify if the results obtained coincide.

ACKNOWLEDGMENT

Our appreciations to the Master's Degree Software Engineering Program from the Computing and Informatics area in Universidad de Tarapacá. Likewise, our special appreciations to Centro Dedicado de Desarrollo del Banco de Crédito del Peru, whose staff collaborated by providing information and relevant data for this research.

REFERENCES

[1] L. Londoño, R. Anaya, M. Tabares, "Análisis de la Ingeniería de Requisitos Orientada por Aspectos según la industria del Software". Revista EIA. Volumen 9. ISSN 1794-1237, pp. 43-52. July 2008.

[2] K. J. Cobsilen, C. J. P. Tablang, & L. Ruiz, "Rational Unified Process (RUP)". SCSIT Research Journal, 1(1), 2012.

[3] C. Lopez, K. Lieberherr, "AP/S++: Case-study of a MOP for purposes of software evolution". Xerox PARC and Northeastern University. San Francisco, USA, 1996.

[4] A. Villazon, W. Binder, & P. Moret, "Flexible calling context reification for aspect-oriented programming". In Proceedings of the 8th ACM international conference on Aspect-oriented software development (pp. 63-74), ACM, 2009.

[5] A. Przybyłek, "Separation of Crosscutting Concerns at the Design Level: an Extension to the UML Metamodel". Department of Business Informatics, Gdańsk University. Gdańsk, Poland, 2008.

[6] J. C. Morocho, J. Chuico, y W. Vásquez, "Denfinición e implementación de los lineamientos para el paradigma de programación orientada a aspectos, en el desarrollo de software". Loja - Ecuador : Universidad Técnica Particular de Loja, 2007.

[7] Casas, Sandra, y otros, "Conflictos entre Aspectos en Etapas del Desarrollo de Software". Rio Gallego - Argentina : Universidad Nacional de la Patagonia Austral, 2007.

[8] S. Apel. "The Role of Features and Aspects in Software Development". PhD thesis, School of Computer Science, University of Magdeburg, 2007.

[9] A. Przybyłek, "Separation of Crosscutting Concerns at the Design Level: an Extension to the UML Metamodel". Gdańsk - Polonia : Department of Business Informatics, Gdańsk University, 2008.

[10] A. Rashid, T. Cottenier, P. Greenwood, R. Chitchyan, R. Meunier, R. Coelho, M. Sudholt, W. Joosen, "Aspect-oriented software development in practice: Tales from aosd-europe". Computer, 43(2), 19-26, 2010.

[11] L. Fuentes, P. Sanchez. "Towards Executable Aspect-Oriented UML Models". 10th International Workshop on Aspect-Oriented Modeling at AOSD'07. Vancouver, Canada, 2007.

[12] Kumar, P. "Aspect-oriented software quality Model: The AOSQ Model". Advanced Computing: An International Journal (ACIJ), 3(2), 2012.

[13] S. Apel, & C. Kästner, "An Overview of Feature-Oriented Software Development". Journal of Object Technology, 8(5), 49-84, 2009.

[14] M. S., Ali, M., Ali Babar, L., Chen, & K. J. Stol, "A systematic review of comparative evidence of aspect-oriented programming". Information and software Technology, 52(9), 871-887, 2010.