Evolving a Software Products Line for E-commerce Systems: a Case Study

Raphael P. Azzolini
Institute of Computing
University of Campinas
Campinas, Brazil
ra144657@students.ic.unicamp.br

Cecília M. F. Rubira
Institute of Computing
University of Campinas
Campinas, Brazil
cmrubira@ic.unicamp.br

Leonardo P. Tizzei
IBM Research
São Paulo, SP, Brazil
ltizzei@br.ibm.com

Felipe N. Gaia
Federal Institute of SP
Boituva, Brazil
felipegaia@ifsp.edu.br

Leonardo Montecchi
University of Florence
Firenze, Italy
leonardo.montecchi@unifi.it

ABSTRACT

Software Product Lines engineering is a technique that explores systematic reuse of software artifacts in large scale to implement applications that share a common domain and have some customized features. For improving Product Line Architecture evolution, it is advisable to develop Software Product Lines using a modular structure. This demand can be satisfied by an aspect-oriented and component-based feature-architecture method that integrates components, aspects and variation point aspect-connectors. This approach allows minimization of feature scattering in the architectural model and supports modular modelling of crosscutting features. A case study mapping major features of significant e-commerce systems operating in Brazil and other countries was performed to evaluate this approach. The assessment of our solution was performed comparing its stability and modularity with other two approaches. Our results indicate that change impact in the architectural model is reduced when using our solution in the context of Software Product Lines evolution.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architecture; D.2.13 [**Software Engineering**]: Reusable Software

Keywords

Software product lines, e-commerce, software architecture stability, component-based development, aspect-oriented development, software evolution

1. INTRODUCTION

Clements and Northrop [5] conceived Software Products Lines (SPL) as a set of software systems that share common domain and have distinct features in order to satisfy a market's segment. This concept promotes software reuse in large scale, providing ways to develop customized products at a low cost. Despite its advantages, SPL evolution can be impaired by the inefficiency of variability mechanisms in accommodating changes, especially those affecting its Product Line Architecture (PLA). Architectural stability means that the PLA can endure evolutionary changes by sustaining its modularity properties.

Component-based development (CBD) [20] and aspect-oriented development (AOD) [15] are techniques that can be used to support SPL's evolution. CBD is a structuring technique where systems are developed by means of software components, providing separation between specification and implementation, favouring modularization and software reuse. AOD is a technique that uses aspects, which are modular units for supporting the encapsulation of crosscutting features by means of composition mechanisms, such as pointcut-advice and inter-type declarations [9]. Feature is a domain property or a visible characteristic to the final user. Crosscutting features represents concerns that are widely-scoped properties and usually crosscut several modules in the software system, which may cause large impact in evolving the PLA.

Tizzei *et al.* [22] have proposed the use of variation point aspect-connectors, called VP-connectors, to support PLA stability in order to modularize crosscutting concerns. The VP-connector solution was evaluated in a small case study comparing two different implementations of the same SPL. The first implementation applied the concepts of VP-connector, aspect (AO) and component (CB) while the second implementation used a pure-component approach (i.e., without aspects and VP-connectors). As a consequence, the study could not identify individual benefits of applying VP-connectors when compared to an implementation based only on aspects and components. The authors also proposed a solution called AO-CB feature-architecture method, ACFAM for short, for mapping crosscutting features identified in the feature model to architectural aspect-elements [23]. The concept of AO-feature models was proposed by extending the feature model proposed by Kang *et al.* [14] and the ACFAM approach includes an AO feature-modelling that extends the FArM method [19] with crosscutting concerns.

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia

DOI: http://dx.doi.org/10.1145/2797433.2797460

The claim is that ACFAM solution can facilitate evolution of PLAs; however, further studies are necessary to investigate to what extent the impact of explicit encapsulation of variation points by means of VP-connectors can facilitate SPL evolution when applied to real complex and large applications.

In order to thoroughly evaluate the ACFAM solution by means of a case study based on a real industry context, we have chosen E-commerce systems, since they play an important economic role on Information Technology (IT) market growth. In 2014, the expectations were that worldwide e-commerce or Business-to-consumer (B2C) sales increase by nearly 20%, reaching $1.471 trillion of dollars [8]. In this domain, integration with other systems is commons, as for shipping and payment services, for which high level of security is important. The system needs to be protected from unauthorized access to customers personal data or payment service. In addition, the system demands an efficient user experience for costumers trust in its services [6].

The main contribution of this paper is to evaluate the ACFAM approach by developing an open source e-commerce SPL, called Mercurius-SPL, available at a public repository [11]. Our case study consists of developing and comparing three different SPL implementations: a component-based implementation, called CBImpl, a pure-aspect-component implementation, called AO-CBImpl, and a VP-AO-CB implementation, called VP-AO-CBImpl. Each implementation has four releases, in a total of 12 releases. Each release includes evolution scenarios for including mandatory and non-mandatory (optional and alternative) features in the e-commerce SPL. We used metrics suites for change impact (lines of code and operations added and modified) [25] and modularity (average coupling, separation, interlacing of features) [4] to measure the architecture stability evaluation of the 3 implementations.

The results pointed out that VP-AO-CBImpl tends to promote better PLA resilience when compared to the other two approaches involved in our study. Our findings confirm the preliminary results of Tizzei *et al.*. Moreover, our study highlights the individual impact of aspects and VP-connectors in achieving PLA stability.

This paper is structured as follows. Section 2 presents some necessary concepts to understand the rest of this paper. Section 3 presents the e-commerce domain study and Mercurius-SPL implementation. Section 4 presents the results of impact of changes and modularity analysis. In Section 5 we discuss the results of this work. Section 6 lists some limitations of our study. Section 7 discuss some related work, and finally, in Section 8, we draw our conclusions.

2. BACKGROUND

2.1 Aspect-Oriented Development (AOD)

Aspect-oriented development (AOD) separates concerns in modules called aspects. Concern is anything a stakeholder may want to consider as a conceptual unit of the system [18], and crosscutting concerns are those that affect multiple modules of a software system. Persistence objects, access security, concurrent access are some examples of crosscutting concerns. This kind of concern causes scattering in the architecture, hindering new features to be added and removed in the system. This problem can be avoided using aspects to modularize crosscutting concerns. These concerns are concentrated in a aspect, that injects them in the parts of code that need them, favouring the code modularity.

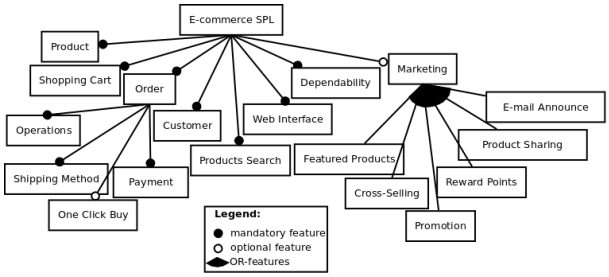

Figure 1: Partial feature model for e-commerce systems

Crosscutting Programming Interfaces (XPI) [12] improve the reuse and modularity of a system mediating the relationships and interactions between classes and aspects, abstracting an existing crosscutting behaviour in the code.

It operates in a manner analogous to the concept of API, separating the specification from the implementation. In other words, the implemented advices do not know the joint points that will be crosscut, which are defined by the XPI, providing higher decoupling for the SPL.

2.2 AO-CB Feature-Architecture Method (ACFAM)

2.2.1 AO Feature Method

Kang *et al.* [14] define features as system attributes that affect directly the final user. These features are documented in a model, like the one in the Figure 1, that represents the hierarchy, composition rules and their rational analysis. A feature model is used for the representation of the SPL variability and is built with mandatory, optional, alternative and OR-feature features [7].

The feature model can be transformed in the architectural model of the SPL. The transformation of the features model of the Figure 1 was developed according to the AO Feature Method [23].

First, the features model has been transformed into the PLA model. To do that, AO Feature Method specifies four transformation steps: **T1** Remove features not related with the architecture and resolve the non-functional features; **T2**: Transform based on the architecture requirements; **T3**: Transform based on the features interactive relations; and **T4**: Transform based on the hierarchical relationships.

These steps will transform features into elements, classes or methods, resulting in the initial architectural model. To obtain the architecture with variation point connector, the crosscutting interfaces and crosscutting connectors have to be specified. The AO Feature Method gives the following steps for specifying them: **S1**: Identify base and crosscutting interfaces; **S2**: Specify base and crosscutting interfaces operations; **S3**: Assess legacy components; **S4**: Implement/refactor the base and crosscutting components; and **S5**: Specify and implement base and crosscutting connectors.

2.2.2 Architectural model with VP-connectors

The result from this methodology is the Component System Model for Software Architectures with Variation Points

(VP-AO-CB model) [22]. This model is an extension of the Component System Model for Software Architectures [10], which provides guidelines to realize architectural elements and explicitly separates the specification and the components implementation [10]. The component specification defines its services through provided interfaces and its dependencies of other services through required interfaces. The component implementation is encapsulated by restricting external access to prevent unwanted dependencies between components.

The architectural model with VP-connector extends the Component System Model for Software Architectures providing architectural elements (components and connectors) that can be base (component elements) or crosscutting elements. Crosscutting components use aspects to modularize crosscutting features. Also, they are responsible for modularizing the variants in order to favour architecture stability. VP-connectors aim to minimize scattering of architectural variation points, providing guidance on how to implement them. They give the necessary treatment to the provided interfaces of the crosscuting modules and crosscuts provided XPI of modules affected by them. The provided XPI is responsible for deciding the join points of the connector's advices.

3. CASE STUDY: THE MERCURIUS-SPL FOR E-COMMERCE

The goal of this case study is to answer the following research questions: **RQ1** can VP-AO-CB model can be applied for a real application such as the e-commerce domain? **RQ2** is VP-AO-CBImpl more effective for the e-commerce PLA stability than CBImpl and AO-CBImpl? **RQ3** what are the individual contributions of AOD and VP-connector for the PLA maintenance and stability?

To answer **RQ1** we identified the e-commerce domain features and used the methodology of described in Section 2.2 to transform then into the PLA. For answering **RQ2**, we made quantitative analysis of the implementations in four different releases, comparing and assessing them to find out which one has better results for evolving the PLA. Our hypothesis was that VP-AO-CB approach contributes for lower change impact and higher modularity than the other implementations. The difference between the three presented implementations is the answer for **RQ3**.

3.1 The E-commerce Domain

The first step of our case study was to extract the e-commerce domain features. They were extracted from four significant e-commerce systems operating in Brazil and USA: a Brazilian store that sells many kinds of products; a Brazilian store specialized in female fashion products; a USA store that sells many kinds of products; and a USA store specialized in supplements and health products. We did not ask permission to divulge the stores names therefore they are omitted in this work. 84 features were extracted from these systems. We identified common and variable features according to the type of product sold, such as attributes like size or weight, and the system country, such as payment method and product devolution rules.

The e-commerce has a set of similar software systems sharing common features that can be reused, it also has distinct features that make them different from each other [16]. In many cases, the need to customize one e-commerce system makes necessary the development of a new product without the software reuse, that can be hindered by inefficiency of variability mechanisms.

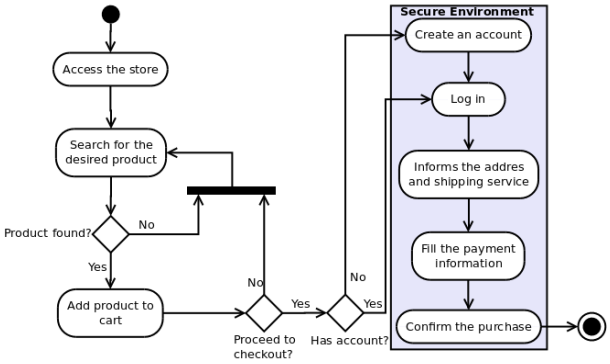

Figure 2: E-commerce buying process

The buying process in an e-commerce system, described in Figure 2, is the same or similar in every kind of online stores in the Internet. It is important that all e-commerce systems have a secure environment at least in the operations marked in the Figure 2.

The payment process is mediated by a payment gateway, responsible for communication between the bank that receives the customer's payment and the store's bank. The e-commerce system has to communicate with the payment gateway by means of a webservice. First, the system sends the payment information to the gateway; secondly, the payment gateway informs that it received the information; and finally, when the payment is confirmed or cancelled, the payment gateway makes the necessary bank transactions and informs the e-commerce system about the confirmation or cancellation of the order's payment.

3.2 The SPL Design

We used the ACFAM methodology to build a feature model and transform it into the PLA. A simplified model is presented in Figure 1.

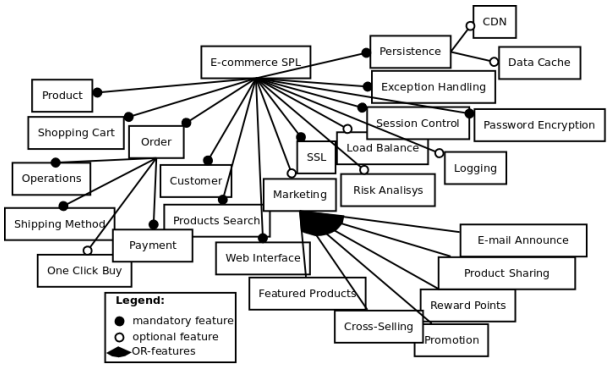

Figure 3: Partial feature model after transformations

Following the transformation **T1** of Section 2.2, dependability and its subfeatures were transformed into functional features (risk analysis, password encryption, SSL, session

control and load balance). Then, in **T2**, features related with the architecture requirement were added to model, they are: logging, persistence, and exception handling. Figure 3 shows the feature model after these two transformations.

In **T3** and **T4**, the relationship and hierarchy between the the features were defined, resulting in the initial architectural model presented in Figure 4.

Following the specialization steps defined by the ACFAM method, the interfaces described on **S1** were defined and specified as described in **S2**. Figure 4 presents the architecture obtained after these steps. The refinement steps of Sections **S3** and **S4** were not executed because we did not have legacy components.

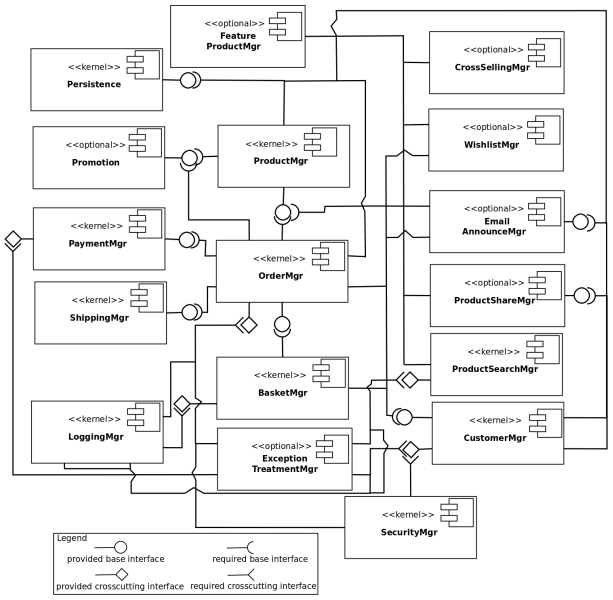

Figure 4: PLA with its base and crosscutting interfaces

For each relation between the components of Figure 4, one connector was specified. As defined in **S5**, these connectors had to be specified as base or crosscutting connectors. Base connectors were implemented where the relation between components is realized by a base interface and crosscutting connectors were implemented where the relation between components is realized by a crosscutting interface.

3.3 Evolution Scenarios

After transforming the feature model into the PLA model and specifying the interfaces, we obtained a PLA for the VP-AO-CB model. This model is composed by the specifications of the interfaces and connectors from the architectural modules and how they are connected. To evaluate this model it was necessary to develop it in code and compare against other possible implementations, derived from the PLA in Figure 4, for the e-commerce SPL, hence, as described in Table 1, three implementations were developed: CBImpl, AO-CBImpl, and VP-AO-CBImpl. These approaches were chosen because with them it is possible to isolate individual contributions of each technique used in the VP-AO-CB model. This development was carried out with Java and AspectJ [21].

Each implementation has four releases: the first release implements the shopping flowchart from the Figure 2; in the Release 2 the exception handling feature was implemented; in the Release 3 the discount promotion feature was implemented; finally, in the Release 4, the data cache feature was implemented. Table 2 shows the summary of the releases in Mercurius-SPL.

Impl	Type	Techniques
I1	component-based implementation (CBImpl)	component-based development
I2	aspect-component implementation (AO-CBImpl)	aspect-oriented programming, component-based development
I3	VP-AO-CB implementation (VP-AO-CBImpl)	VP-connector, aspect-oriented programming, component-based development

Table 1: Case study implementations

Release	Description	Type of Change
R1	E-commerce core	
R2	Exception handling included	Inclusion of mandatory crosscutting feature
R3	Discount promotion included	Inclusion of optional crosscutting feature
R4	Data cache included	Inclusion of optional crosscutting feature

Table 2: Summary of the releases for the Mercurius-SPL

3.3.1 Release 1

The main objective of this release is to create e-commerce systems where it is possible to sign-in, search for a product and buy it. Hence the features implemented in this release were: product, product attributes and description, physical product type, category, shopping cart, order, credit card payment method, customer, customer address, customer sign-in and login, password encryption, product search by text, persistence, and logging.

In our solution, the password encryption and logging features were implemented with aspects and VP-connector. The first one crosscuts customer login and registration methods; the second one crosscuts methods that should have log for analysis.

3.3.2 Release 2

In this release we implemented exception handling, a mandatory and crosscutting feature. This feature was chosen because it allow us to evaluate how the stability of the PLA is affected by adding one mandatory feature and because its modularization by means of AOD promotes reuse [1]. For implementing this feature in the VP-AO-CBImpl we used the method proposed by Iizuka *et al.* [13], that uses the VP-connector for exception handling.

3.3.3 Release 3

In this release was added the discount promotion feature. This optional feature was chosen because it is a default promotion feature used by all the assessed e-commerce systems.

Furthermore, this features crosscuts main features of the e-commerce domain, such as products, products search and order.

With this feature, the products can have promotional discounts, in percent of the product value or a fixed value. In our solution, it was implemented a VP-connector that enables to connect another types of promotion.

3.3.4 Release 4

This is the release where the cache feature was added. This feature crosscuts the persistence feature and is responsible to store accessed data from the database in memory for a certain amount of time, as a way to improve the system performance since disk access is much more costly than memory access. The fact that this feature crosscuts the persistence feature, that is present in almost all modules of the SPL, justifies the need of assessing this change.

4. STUDY RESULTS

This section presents the results of an analysis made comparing the three SPL implementations. Two assessment procedures were made: a change impact analysis, where we made a quantitative analysis of typical change impact measures [25], such as the number of lines of code, the number of operations inserted or modified; and a analysis of the modularity of the SPL throughout the implemented changes, where we choose metrics already used in other works [9][3] for the same purpose.

The assessment was made in the four releases of each implementation presented in the Section 3.3: Release 1 (**R1**); Release 2 (**R2**); Release 3 (**R3**); and Release 4 (**R4**).

4.1 Change Impact Analysis

4.1.1 Lines of Code

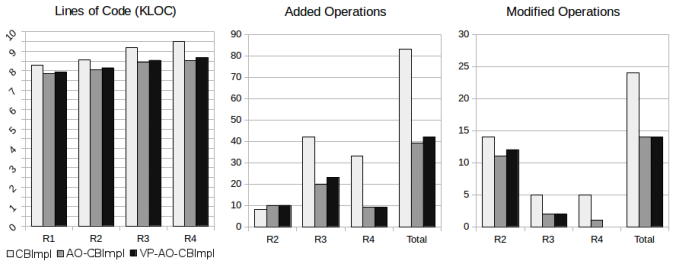

Figure 5: Thousands of lines of code, operations added and modified in the SPLs

For each implementation we measured the number of the Kilo of lines of code (KLOC), where 1 KLOC = 1000 lines of code. The results are in the Figure 5, where each group of bars represents one of the implemented releases and each bar represents one of the SPLs.

Even in the first release, where the KLOC of the SPLs have a small difference, about 250 lines of code, the pure-component implementation always have the higher KLOC. Moreover the increase of lines of code in the pure-component is higher than in the other implementations, in R4, the difference of KLOC between the pure-component and the implementation with the variation point connector jumps from 250 lines of code to 1000 lines of code.

4.1.2 Operations Added and Modified

Figure 5 also shows the results of the metrics for the number of operations added and modified. Except R2, which, unlike the other two SPL, the pure-component implementation had no exception handling module added and was naturally added a lower number of operations, the number of new operations and operations to be modified in each change in SPL is always higher in the pure-component implementation.

4.2 Modularity Analysis

4.2.1 Average coupling between modules

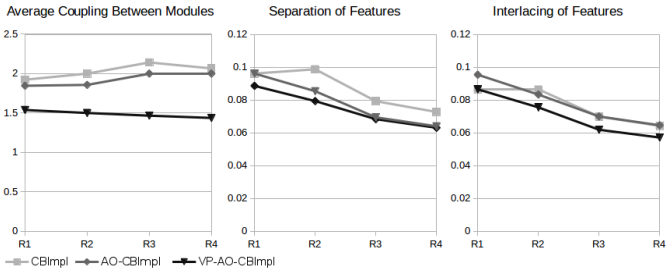

Figure 6: Average coupling, separation and interlacing of features in the modules throughout the changes in the SPL

Average coupling between modules shows the dependency that they have on each other, a lower coupling means that the modules have a higher modularity. We calculated the average coupling between modules based on the metrics for coupling between objects given by Chidamber and Kemerer [4]. The arithmetic mean of the coupling of all modules in the SPL was calculated, where the coupling of one module is the number of modules it depends on.

As shown in the Figure 6, the average coupling of the three implementations was stable after the releases changes. However, the average coupling between the modules of the implementation with the variation point connector is lower than the other implementations.

4.2.2 Separation of features

Separation of features shows how modularized the features are in the SPL. We used the metrics of Riebisch and Brcina [17] to calculate the separation of each features. The lower is the value calculated the lower is the scattering of features among the modules of the SPL.

In the Figure 6 is shown that the components with aspects and the components with aspects plus variation point connectors implementations have similar values of separation of features and, as these values are lower than the pure-component implementation, the aspects utilization promotes higher modularization of the SPL's architecture.

4.2.3 Interlacing of Features

Interlacing of features shows how many features the same module implements. As we did made for the separation of features, to calculate the interlacing of features we used the metrics of Riebisch and Brcina [17]. The lower is the calculated value the higher is the SPL modularization and the easier is to evolve the PLA.

The implementation based on components, aspects and variation point connector has lower interlacing than the other two implementations, that have similar interlacing between each other.

5. DISCUSSION

In our case study, Mercurius-SPL was successfully implemented with the ACFAM solution, answering **RQ1** and showing evidences that the model can be applied for this domain. Furthermore, answering **RQ2**, after assessing the collected data, we conclude that this solution is more effective for the e-commerce PLA stability than the other approaches, as changes in code are lower and the architecture is more stable in our solution. The answer for **RQ3** is as follows.

Aspects contributes to reduce the efforts for changing and maintaining the code. In our case study, AO-CBImpl and VP-AO-CBImpl presented lower KLOC than CBImpl. Moreover, the increase of KLOC from one release to another in CBImpl is higher than in AO implementations, as the difference jumps from about 250 lines of code in the first release to about 1000 lines of code in the last release. The number of operations added and modified reinforce this observation, as CBImpl needs to modify more operations than the AO implementations in order to add new features.

VP-connector and XPI interfaces promotes higher modularization of the SPL's architecture. Despite AOD contributes to a lower separation of features, in our study it could not present better results than a pure-component implementation for interlacing of features and can harm the coupling between modules. However, when using VP-connector, in the VP-AO-CBImpl, the number of these metrics was reduced, presenting satisfactory results.

The VP-connectors model adds the benefits of the AOD for reducing the impact of changes to the PLA, contributing for gains in stability. Moreover, it contributes to nullify possible harm that AOD can give to the modularity of the SPL, further increasing its architecture evolution stability.

6. THREATS TO VALIDITY

The validity of the results is closely linked to how well their threats have been addressed. Considering the Wohlin [24] classification, four threats are mapped: conclusion validity, internal validity, external validity and construct validity.

Conclusion validity concerns the relationship between treatment and outcome. It means that the conclusions based on the statistical analysis of the data are significant. To mitigate the threat we used metrics that have already been used in other works [25][9][3] for SPL evolution stability assessment. A sample of both type of data were verified to make sure that the process had not errors.

Internal validity refers to as an independent variable can be affected, in other words, if a causal relationship exists between treatment and outcome. A possible threat in this study is the fact that the releases of the implementations were developed by the first author of this work. To mitigate this threat, systematic methods, reported on the literature [19][23], were applied starting from the same requirements in order to develop all the implementations. The requirements were based on representative e-commerce systems and first author's know-how, acquired from three years working in the e-commerce industry, also helped to reduce the influences in results.

External validity is concerned with generalization of the results. We chose to use programming language and tools already used in other works [9][22] for the same purpose of evaluating SPL evolution stability.

Finally, concerning the construct validity, the modularity metrics were collected manually, which can cause error in the measures. To mitigate the threat the collected data were double checked.

7. RELATED WORK

Laguna and Hernandez [16] present an e-commerce SPL development for the .NET platform. Unlike Mercurius-SPL, their approach does not use components and aspects techniques for implementing the SPL.

Figueiredo *et al.* [9] assessed the positive and negative impacts of AOD for supporting the PLA evolution stability. They concluded that AOD copes well with the separation of features with no shared code and adheres better than object-oriented programming languages to well-known design principles. However, they identified that this strategy is vulnerable to changes targeting core features.

Tizzei *et al.* [22] presents the study of the PLA stability with the VP-AO-CB. However, their solution do not present the development of a real and relatively large application and did not evaluate individual contributions of VP-connector and aspects for PLA stability.

8. CONCLUSIONS AND FUTURE WORK

This paper presented a real case study to evaluate the ACFAM solution for implementing SPL for the e-commerce domain, that is widely used in industry. The result of this case study was the Mercurius-SPL, an open source e-commerce SPL of a modular PLA, resistant to changes. We compared Mercurius-SPL implementation (VP-AO-CBImpl) with CBImpl and AO-CBImpl, assessing its stability and modularity.

We concluded that the impact of changes in the architecture is reduced when using aspects in the SPL, as showed in the results of the Section 4. Another contribution of the use of aspects is the lower scattering of features, favouring higher modularization these implementations. Moreover, the use of VP-connector contributes to nullify possible harm that AOD can give to the PLA stability, favouring lower interlacing between modules, contributing to the maintenance and evolution of PLA.

For future work, we will continue the development of the remaining features and improve the study of the impact of non-functional requirements associated with crosscutting features in the PLA stability. Furthermore, we are studying ways to automate the ACFAM method, facilitating and optimizing the process of transforming and specializing the feature model into the architectural model.

Acknowledgements

This work has been partially supported by the DEVASSES project, funded by European Union's seventh framework programme under grant agreement PIRSES-GA-2013-612569 and by the AMADEOS project [2] under grant agreement No. 610535.

9. REFERENCES

[1] A. Almeida, E. Barreiros, J. Saraiva, F. Castor, and S. Soares. Is exception handling a reusable aspect? In *SBCARS 2014*, pages 32–41. IEEE, 2014.

[2] *AMADEOS:* Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems, 2013. Seventh Framework Programme, FP7-ICT-2013-10.

[3] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena. Composing design patterns: a scalability study of aspect-oriented programming. In *International conference on Aspect-oriented software development*, pages 109–121. ACM, 2006.

[4] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[5] P. Clements and L. Northrop. *Software product lines: practices and patterns*, volume 59. Addison-Wesley Reading, 2002.

[6] B. J. Corbitt, T. Thanasankit, and H. Yi. Trust and e-commerce: a study of consumer perceptions. *Electronic commerce research and applications*, 2(3):203–215, 2003.

[7] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[8] eMarketer. Worldwide Ecommerce Sales to Increase Nearly 20% in 2014. Available at <http://www.emarketer.com/Article/Worldwide-Ecommerce-Sales-Increase-Nearly-20-2014/1011039>. Accessed in 30 Dec. 2014, 2014.

[9] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, and F. Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *International Conference on Software Engineering*, pages 261–270, NY, USA, 2008. ACM.

[10] L. A. Gayard, C. M. F. Rubira, and P. A. de Castro Guerra. Cosmos*: a component system model for software architectures. *Tec. Rep. IC-08-04, Instituto de Computaçao*, 2008.

[11] GitHub - Mercurius. Available at <https://github.com/raphaelazzolini/mercurius>. Accessed in 24 apr. 2015.

[12] W. Griswold, M. Shonle, K. Sullivan, Y. Song, N. Tewari, Y. Cai, and H. Rajan. Modular software design with crosscutting interfaces. *Software, IEEE*, 23(1):51–60, Jan 2006.

[13] B. Iizuka, A. S. Nascimento, L. P. Tizzei, and C. M. Rubira. Supporting the evolution of exception handling in component-based product line architecture. In *Exception Handling, International Workshop on*, pages 62–64. IEEE, 2012.

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990.

[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 - Object-oriented programming*, pages 220–242. Springer, 1997.

[16] M. A. Laguna and C. Hernandez. A software product line approach for e-commerce systems. *International Conference on e-Business Engineering*, 0:230–235, 2010.

[17] M. Riebisch and R. Brcina. Optimizing design for variability using traceability links. In *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pages 235–244. IEEE, 2008.

[18] M. P. Robillard and G. C. Murphy. Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, 16(1), 2007.

[19] P. Sochos, M. Riebisch, and I. Philippow. The feature-architecture mapping (farm) method for feature-oriented development of software product lines. In *Engineering of Computer Based Systems,13th Annual International Symposium and Workshop on*, pages 9 pp.–318, 2006.

[20] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[21] The AspectJ Project. Available at <http://www.eclipse.org/aspectj/>. Accessed in 26 jul. 2014.

[22] L. P. Tizzei and C. M. Rubira. Aspect-connectors to support the evolution of component-based product line architectures: a comparative study. In *Software Architecture*, pages 59–66. Springer, 2011.

[23] L. P. Tizzei, C. M. Rubira, and J. Lee. An aspect-based feature model for architecting component product lines. In *Software Engineering and Advanced Applications, EUROMICRO Conference on*, pages 85–92. IEEE, 2012.

[24] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen. Experimentation in software engineering: an introduction. 2000, 2000.

[25] S. Yau and J. Collofello. Design stability measures for software maintenance. *Software Engineering, IEEE Transactions on*, SE-11(9):849–856, Sept 1985.