Identifying Competitive Interaction of Patterns in Software Product Lines

Identificación de la interacción competitiva de patrones en líneas de producto software

COLCIENCIAS TIPO 4. ARTÍCULO CORTO
RECIBIDO: JUNIO 1, 2015; ACEPTADO: JUNIO 28, 2015

Julián Cifuentes
jacifuentes@icesi.edu.co
Hugo Arboleda
hfarboleda@icesi.edu.co

Universidad Icesi, Cali-Colombia

Abstract

Enterprise design patterns (such as JEE patterns) can be used to promote Quality Attributes (QA) as functional features when deriving products in an Enterprise Applications Software Product Line (SPL). One of the issues found in product derivation is the interaction of code fragments generated by applying patterns. This interaction can be collaborative or competitive. When competitive, relationships can be adaptable or excluding. In both cases, different approaches (e.g., pattern composition, pattern substitution, constraint reasoning) can be used to address the problem. However, identifying and predicting these interactions early can be useful to develop suitable strategies. In this work, we explore and identify competitive feature interactions using two base repositories: QAs from a known standard and patterns from a catalogue. We show two cases of feature interaction, when promoting specific levels of QAs in an Enterprise Application SPL.

Keywords
Software product lines; quality attributes; feature interaction.

Resumen

Los patrones de diseño de software pueden ser utilizados para la promoción de Atributos de Calidad (QA) como características funcionales durante la generación de productos en una línea de productos de software (SPL) de aplicaciones empresariales. Uno de los problemas que se presentan en la derivación de productos es la interacción de los fragmentos de código derivados por los patrones aplicados. Esta interacción puede ser de naturaleza colaborativa o competitiva. Cuando es competitiva, las relaciones pueden ser adaptables o excluyentes. En ambos casos es posible utilizar diferentes enfoques para abordar el problema (e.g., composición de patrones, sustitución de patrones, razonamiento basado en restricciones). La identificación y predicción temprana de estas interacciones puede ser útil para desarrollar estrategias adecuadas. En este trabajo, exploramos e identificamos interacciones competitivas usando dos repositorios base: un estándar conocido de calidad que define Atributos de Calidad y un catálogo de patrones de diseño. Como resultado mostramos dos casos de interacción cuando se promueven niveles específicos de QA en una SPL de aplicaciones empresariales.

Palabras clave
Línea de productos de software; atributos de calidad; interacción funcional.

I. INTRODUCTION

Software Product Line Engineering [SPLE] is an expanding approach in software engineering, which aims at developing a set of software systems that share common features and satisfy the requirements of a specific domain (Pohl, Böckle, & van der Linden, 2005). SPLE puts strong emphasis on "re-use" organized through common software architecture (Arboleda & Royer, 2012). A software product line (SPL) is a family of related software products sharing a common set of assets. Differences and commonalities between products are typically described in terms of features. Users can specify a product as a selection of features that satisfies their functional requirements. With many contemporary implementation mechanisms, it is possible to automatically generate products based on feature selections (Siegmund et al., 2013; Durán & Arboleda, 2014).

One of the challenges in today's both traditional software engineering and SPLE approaches is to deliver high-quality software on time to customers and software quality has become a major concern of software organizations (Wieczorek & Meyerhoff, 2001). A lot of research about Software quality has been done to refine the concept of quality into a number of quality attributes [QAs], also known as quality characteristics, quality factors or non-functional requirements [NFRs] (Anderson, Bajaj, & Gorr, 2002).

There are studies that deal with the existence and influence of QAs on the analysis and design stages (Myllärniemi, Männistö, & Raatikainen, 2006; Halmans & Pohl, 2003) and explicitly consider QAs variations and their relationships with functional features. Other works like Durán y Arboleda (2014) and Hallsteinsen, Fægri, and Syrstad, (2004), have explored how to promote quality attributes as functional features using enterprise design patterns.

Regarding the relationships between couples formed by QAs and Design Patterns, these can be synergistic or competitive depending of its interaction. When competitive, relationships are classified as adaptable or exclusionary. Adaptable means that patterns can be composed and adapted to achieve the required quality scenarios. Exclusionary means that patterns cannot coexist. Different approaches (e.g., pattern composition, pattern substitution, constraint reasoning) can be applied to resolve exclusionary relationships between design patterns when the expected quality attributes they promote are all required (Benavides, Trinidad, & Ruiz-Cortés, 2005; Nhlabatsi, Laney & Nuseibeh, 2008). Identification of competitive interactions at early stages of SPLE is an essential issue because it helps to prevent subsequent failures in products of the SPL (Laney & Nuseibeh, 2008).

In this paper we consider two repositories (one for QAs and another for design patterns) and demonstrate how these repositories can be helpful to define a systematic way of classifying and identifying feature interactions. Such interactions can be produced e.g., when design patterns promote desired levels of QAs, previously selected in a variability model attached to a SPL of enterprise applications (Durán & Arboleda, 2014).

II. METHODOLOGY

A. Identifying Quality Attributes

Regarding the Quality Attributes, we have considered the definition of Quality Attributes given in the ISO 25010 standard (2011), which describes a software product quality model that categorizes software quality in eight characteristics which are further subdivided into subcharacteristics and attributes (Figure 1). This model is a comprehensive specification that provides a set of quality characteristics and quality attributes that determine functional features on the system.

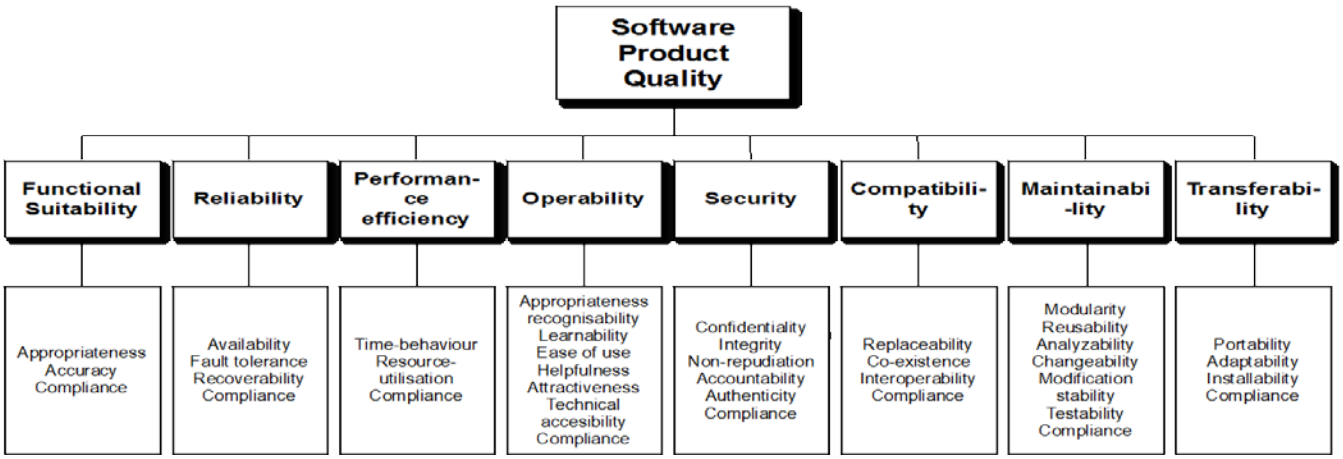
B. Identifying Design Patterns

As a design patterns repository we worked with the catalogue of Java EE Patterns defined in Adam Bien's book (2012). As said in the section above, there are works that have demonstrated how to promote functional features determined by quality attributes using specific design patterns. We have considered all of them along our study. However, use of other catalogues has not been discarded when needed.

C. Strategy

Our main goal is to perform an exploratory study that leaves open possibilities for future work in which new patterns and its interaction are implemented and analyzed. In this work, we identified some cases of conflict interaction between design patterns promoting functional features. These patterns and features are determined by QAs that have been selected by an SPL engineer from a QA variability model.

Figure 1. Software Product Quality Model (ISO/IEC, 2011)



Even though the method to find the two cases has been determined and mainly led by our criteria, we have used the two repositories described above as a starting point to correlate QAs and design patterns. For each QA found in ISO 25010's quality model, we describe how it can be promoted or affected by one or more design patterns in Adam Bien's catalogue. In this catalogue, for each specified pattern, there is a section called *Consequences* that describes which QAs are affected by the implementation of the pattern. Such consequences can be classified in order to detect patterns that could present potentially conflicting interactions. Our exploration, however, has not been exhaustive and is limited to obtaining the desired cases.

III. RESULTS

A. Case one

According to ISO 25010's software product quality model (Figure 1), the quality attribute Performance is specialized into Time Behaviour, Resource Utilization and Performance Efficiency Compliance. We have selected the first one (Time Behaviour), which determines *the degree to which the software product provides appropriate response and processing times and throughput rates when performing its function, under stated conditions*. Under the same standard, the quality attribute Reliability is specialized into 4 attributes (Availability, Fault Tolerance, Recoverability and Reliability Compliance). We have considered Availability, which determines *the degree to which a software component is operational and available when required for use* (ISO/IEC, 2011).

Different levels for these attributes (e.g., *Normal*, *Medium*, *High*) can be promoted using specific design patterns, as presented by Durán y Arboleda (2014). For example, a medium level of Time Behaviour can be promoted by the *Fast Lane Reader* (FLR) pattern, which provides an efficient way to access large amount of read-only data. Likewise, a *medium* level of Availability could be promoted using a *spare computing* pattern, which can replace a failed component to achieve a degraded operational level when one server is unavailable. So then, the first pattern (FLR) promotes read-only access to the database, and requires a cache to avoid reprocesses for similar requests to the database. The second pattern, on the other hand, promotes the use of persistent storage to maintain application state and avoid the use of memory, meaning that caches are not allowed. Table 1 summarizes this case.

Table 1. Case one results

Characteristic	Subcharacteristic	Level	Promoting pattern	Pattern interaction
Performance efficiency	Time Behaviour	Medium	FLR	Cache required vs. cache not allowed.
Security	Availability	Medium	Spare computing	

B. *Case two*

As in Case 1, we have selected Time Behaviour from Performance as defined in ISO 25010's software product quality model. We have also considered Confidentiality, (Figure 1) from the characteristic Security that is specialized into six sub characteristics (Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity and Security Compliance). According to ISO 25010, confidentiality determines *the degree to which the software product provides protection from unauthorized disclosure of data or information, whether accidental or deliberate*.

Again, different levels for those attributes (e.g., *Normal*, *Medium*, *High, Encrypted, and Unencrypted*) can be promoted using specific design patterns. For example, a high level of Time Behaviour can be promoted by the *Fast Lane Reader* (FLR) pattern adding a *Paginator* strategy, which states that an entire result set can be divided into smaller chunks. Furthermore, *Encrypted* level of Confidentiality can be addressed using a Password Based Encryption (PBE) strategy based on the Java Security API (Oracle, 2014).

The first pattern has impact in some CRUD operations (e.g., retrieve, listRecords). The second pattern has also impact in CRUD operations, including retrieval information that has to be unencrypted before sending it to a final user or GUI. Hence, a strategy or solution for enabling both levels must be developed. Table 2 summarizes this case.

Table 2. Case two results

Characteristic	Subcharacteristic	Level	Promoting pattern	Pattern interaction
Performance efficiency	Time Behaviour	High	FLR + Paginator	CRUD vs. CRUD with encrypted data
Security	Confidentiality	Encrypted	PBE	

IV. Conclusions and Future Work

In this paper we have presented two cases of feature interaction caused by implementation of design patterns that promote specific levels of QAs in an Enterprise Applications SPL. We have taken advantage of the software product quality model described in ISO 25010 as a repository of QAs, and the catalogue of design patterns outlined by Adam Bien in his book of JEE patterns. This study is a concrete contribution that leaves open the possibility for future works in which additional interactions between new patterns can be analyzed. The highest-level goal is to identify general strategies that resolve exclusive interactions between design patterns when promoting desired quality attributes. This mechanism will be used to extend the quality model and the tool proposed by Durán y Arboleda (2014) in order to provide a mechanism to handle these situations during the derivation of products from a configuration of selected quality attributes. Among the possible solution scenarios we have considered to evaluate the following alternatives: using a new design pattern that integrates two or more QAs which individual promoting patterns are exclusionary; to delegate and compose a design pattern starting from another, to recursively include the desired QAs; and define the problem as constraint-based reasoning where the variables are the desired levels of QAs and the constraints are the dependency relationships between them.

V. References

Anderson, B.B., Bajaj, A., & Gorr, W. (2002). An estimation of the decision models of senior IS managers when evaluating the external quality of organizational software. The Journal of Systems and Software, 61(1), 59-75.

Arboleda, H. & Royer, J-C. (2012). Model-Driven and Software Product Line Engineering. ISTE / John Wyley & Sons.

Benavides, D., Trinidad, P., & Ruiz-Cortés, A. (2005). Automated reasoning on feature models. In Advanced Information Systems Engineering - Lecture Notes in Computer Science, 3520, 491-503

Bien, A. (2012). Real World Java EE Patterns Rethinking Best Practices. Raleigh, NC: lulu.com.

Durán, D. & Arboleda, H. (2914). Quality-driven software product lines [Master's thesis]. Universidad Icesi: Cali, Colombia, 2014. Available: http://bibliotecadigital.icesi.edu.co/biblioteca_digital/handle/10906/7749

Hallsteinsen, S., Fægri, T.E., & Syrstad, M. (2004). Patterns in product family architecture design. In F.J. van der Linden [Ed.], Software Product Family Engineering - Lecture Notes in Computer Science, 3014, 261-268. Berlin-Heidelberg: Springer.

Halmans, G. & Pohl, K. (2003). Communicating the variability of a software-product family to customers. Journal on Software and Systems Modeling, 2(1), 15-36.

International Organization for Standardization / International Electrotechnical Commission [ISO/IEC]. (2011). *ISO/IEC 25010: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Software and quality in use models*. Geneva, Switzerland: ISO/IEC.

Myllärniemi, V., Männistö, T. & Raatikainen, M. (2006). Quality

Attribute Variability within a Software Product Family Architecture. Second International Conference on Quality of Software Architecture QoSA, 2006. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.6690&rep=rep1&type=pdf

Nhlabatsi, A., Laney, R., & Nuseibeh, B. (2008). Feature interaction: The security threat from within software systems. Prog. Informatics, 5, 75-90.

Oracle. (2014). Java Security API. [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/security/package-summary.html

Pohl, K., Böckle, G., & van der Linden, F. (2005). Software Product Line Engineering: Foundations, Principles, and Techniques. Berlin, Germany: Springer.

Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P.G., Apel, S., & Kolesnikov, S.S. (2013). Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. Inf. Softw. Technol., 55(3), 491-507.

Wieczorek, M & Meyerhoff, D. (2001). Software quality: State of the art in management, testing, and tools. Berlin, Germany: Springer.

CURRICULUM

Julián Cifuentes. Ingeniero de Sistemas de la Universidad del Valle, con más de 10 años de experiencia como analista, desarrollador y consultor en proyectos de implantación de sistemas de software llevados a cabo en los sectores de comercio electrónico, telecomunicaciones y servicios domiciliarios (Open International Systems, Empresas Municipales de Cali [Emcali], Empresas Públicas de Medellín [EPM], Cablevisión Argentina, Carvajal S.A). Su experiencia como consultor independiente lo ha llevado liderar y apoyar procesos de integración (Óptima, Gases de Occidente), migración (Óptima, Distromel), pruebas de software (GreenSQA, Emcali, Gases de Occidente, Banco de Occidente) e interventoría (Wasser Colombia, Emcali). Actualmente es investigador y docente, y cursa la Maestría en Informática y Telecomunicaciones en la Universidad Icesi.

Hugo Arboleda. Doctor en Informática de la École des Mines de Nantes, Francia. Tiene un Doctorado en Ingeniería de la Universidad de Los Andes, una Maestría en Sistemas y Computación de la misma universidad y es Ingeniero de Sistemas de la Universidad del Valle. Consultor de empresas nacionales como Coomeva, Carvajal y Promédico en temas de adquisición estratégica de recursos de TI, gerencia de proyectos e ingeniería de software. Parte de su desarrollo profesional lo llevó a cabo en empresas como Open International Systems y Heinsohn Business Technology. Autor de más de 30 publicaciones entre libros, artículos de revista y ponencias auditadas nacional e internacionalmente. Ha sido profesor en el Máster Europeo de Desarrollo de Software Orientado por Objetos y Componentes ofrecido por la École des Mines de Nantes (Francia) y Vrije Universiteit Brussel (Bélgica). Fue profesor de la Universidad de Los Andes y de la Universidad Politécnico Gran Colombiano en Bogotá, y de la Universidad de San Buenaventura en Cali. Actualmente dirige la Maestría en Gestión de Informática y Telecomunicaciones de la Universidad Icesi, donde es profesor de tiempo completo.