Journal of Object Technology
Published by ETH Zurich, Chair of Software Engineering, © JOT 2010
Online at http://www.jot.fm.

REquirements, Aspects and Software Quality: the REASQ model

Isi Castillo[a] Francisca Losavio[b] Alfredo Matteo[b]
Jorgen Boegh[c]

a. Universidad Nacional Experimental Sur del Lago, Venezuela
b. Universidad Central de Venezuela
c. Terma A/S, Vasekær 12, DK-2730 Herlev, Denmark

Abstract Object-oriented analysis and design have been more concerned with system functionality, neglecting non-functional aspects; the result is code which is tangled and difficult to maintain, contradicting main principles of object orientation. Aspect Oriented Software Development (AOSD) proposes the early specification of non-functional requirements. However, a standard and homogenous vision of the AOSD terminology is still missing. The goal of this work is to integrate AOSD concepts, classic requirements engineering notions, and the new standard ISO/IEC 25030 on software quality requirements.The main result of this study is the REASQ (REquirements, Aspects and Software Quality) conceptual model, expressed in UML. All the modelling concepts are formalized into three related ontologies, representing the ambits of aspect-orientation, software quality and requirements engineering. The ontologies can be used as an umbrella to specify quality requirements in aspect-oriented engineering processes.

Keywords Aspects, Concerns, Software quality, Requirements engineering, ISO/IEC 25030, ISO/IEC 25010, Ontology.

1 Introduction

The importance of following a mature development process is relevant to software engineering best practices. The software development process has evolved towards process improvement, by guaranteeing to a certain extent the quality of the resulting software product. However, how to describe and guarantee (have a precise rationale) a quality software product according to this mature process is still unclear. In usual practice, a precise rationale for software development is still missing, and often reduced to just fill up a framework or template. Existing accepted software development methods are driven more by the specification of the system functionality, rather than

considering non-functional aspects, related to the behavior of the system functionality or context. These are considered in general at later stages of development, contributing to the scattering and tangling of the final code and contradicting the maintainability property, claimed by the object-oriented software development paradigm.

The importance of a clear understanding and correct specification of requirements is increased by the complexity of actual software systems. In particular, if they have to respond to concerns such as interoperability, adaptability, availability and security, which go far beyond the main system functionality or service offered by a software component, because they can affect multiple components [6]. Hence, these concerns should be considered very early, at the same time as the main functionality of the system; at present, the requirements engineering discipline is trying to establish a precise rationale to consider these kinds of requirements affecting multiple components. For example, the user identification or login functionality in a commercial portal must be compliant with the organization's access control policy, implying a kind of security issue. However, the user credit card acceptance functionality implies also the intervention of a security mechanism, and both security mechanisms have different implementations. Hence a security concern is entangled with two main functionalities, making it difficult to understand, maintain and reuse the code produced according to this design strategy; security is known as a crosscutting concern [1].

The early aspects research trend considers Aspect Oriented Requirements Engineering (AORE) and Aspect Oriented Architectural Design (AOAD) [12, 14] to study the early identification and handling of crosscutting concerns to improve the quality of final code [5, 6, 12, 14, 33]. On the other hand, in the literature the concepts of aspect and concern are frequently considered synonyms. In AORE, the terms requirement and concern are used indistinctly and the crosscutting concerns are associated indistinctly with the software product high level quality properties or quality requirements, such as availability, reliability and low level quality attributes as response time, in the sense of ISO/IEC 9126-1 [23]. Brito and Moreira [5, 6] propose and refine a generic model at requirements engineering stage, for the early identification, separation, integration and composition of crosscutting concerns as aspects. In particular, in [5] they extend a work already presented by proposing a novel process for composition that introduces the new notions of match point, dominant concern and LOTOS operators to define composition rules. Additionally, they recommend the use of existing catalogues (e.g. the NFR framework) to alleviate both the identification and the specification tasks. In [6] they focus on the integration task, discussing how to compose crosscutting concerns with other concerns. To accomplish this, they introduce the notion of match point (an abstraction of the joint point concept [27]), to compose concerns they propose to first identify match points and then, for each one, define a composition rule. They define concerns as both functional and non-functional requirements and a crosscutting concern as candidate aspect because it may be mapped later into an aspect. Moreover, they indicate that some functional concerns can also be crosscutting concerns, and therefore better handled as candidate aspects. Rosenhainer [33] defines crosscutting requirements as the requirements intercrossing with other requirements. Two techniques (identification through inspection and identification supported by information retrieval techniques) are proposed for early identification and documentation of crosscutting requirements from the general requirement specification, according to an aspect-mining point of view to improve traceability between requirements, crosscutting concerns and artifacts of the development process.

The works discussed above point out the lack of homogeneity in the terminology and

that standards are not taken into account, making difficult in general the understanding of the AOSD (Aspect-Oriented Software Development) paradigm. In this sense, domain modeling with a focus on ontology is proposed towards a common language for reuse [13]. An ontology provides a common vocabulary for specifying requirements for a family of applications in the domain. Another research issue is the identification and specification of requirements and their quality properties [3, 7, 29, 35, 39], to establish the goals to be attained by a quality software product. In Chirinos et al. [8], a conceptual model is proposed to identify quality requirements for software products (RECLAMO), based on the quality views of ISO/IEC 9126-1 [23], presented in Figure 1. This model aims to facilitate the tasks of the requirements engineer. It is used as a tool for quality requirements specification in software architecture design, to establish selection criteria for different architectural solutions. Recently, as part of new ISO/IEC 25030 standard for requirements SQuaRE [7, 21], which will be discussed more deeply in section 3, provides also a classification of requirements sharing common points with RECLAMO.

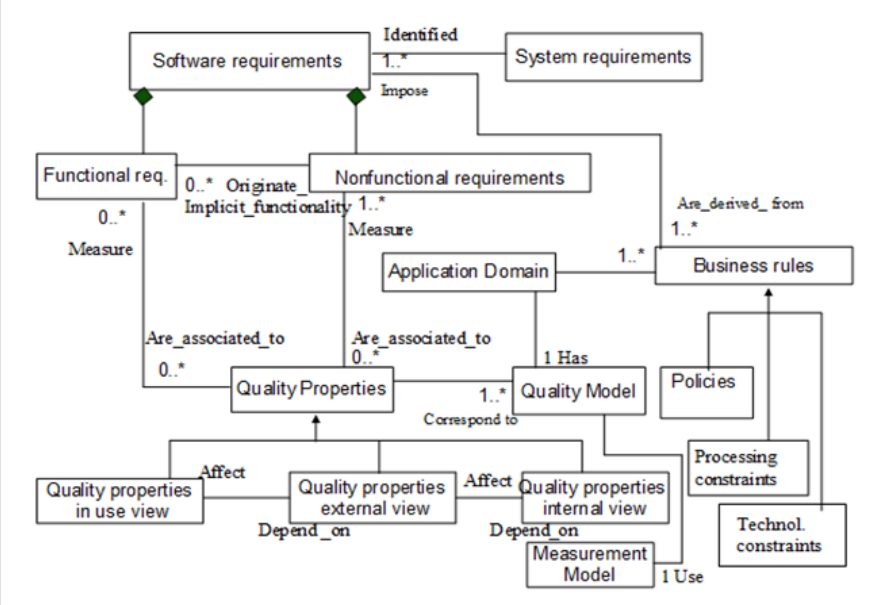


Figure 1 – RECLAMO [8]

The main goal of this work is to establish a unique conceptual model to clarify the AOSD emergent terminology: aspect, composition, (functional, non-functional, cross-cutting) concern, (functional, non-functional) quality or (inherent, assigned) property requirements for the software product. This unified model, called REASQ (REquirements, Aspects and Software Quality), will consider the new ISO/IEC SQuaRE standard [21], integrating ISO/IEC 25030 and ISO/IEC 9126-1 [23] with a requirements classification model proposed in [8]. All the concepts used in the model will be formalized by three related ontologies written in Protégé [18] to facilitate reuse, representing the ambits of aspect-orientation, software quality and requirements engineering respectively; the REASQ model will guide the integration of these ontologies, which will be used as an umbrella to define an aspect-oriented quality requirements engineering process.

This article is structured as follows: Section 2 describes briefly the SQuaRE standards family to specify the software product quality model, in particular the standards ISO/IEC 25030 for software requirements identification, ISO/IEC 25010, and ISO/IEC 9126-1 are discussed. Section 3 presents REASQ as a main result, the conceptual model for the AOSD domain, with the main terms and associations and illustrates a brief instantiation of the REASQ model to show its applicability to a case study. Section 4 uses the REASQ model to define three related ontologies written in Protégé to formalize further our results. Section 5 is dedicated to a brief survey of related works about conceptual modeling in AOSD and the use of quality standards for requirements specification. Finally the conclusion and future perspectives are presented.

2 The SQuaRE ISO/IEC Standard

SQuaRE consists of a family of standards under the general title Software Product Quality Requirements and Evaluation (Figure 2 illustrates the organization of these families or divisions) [21, 22]. The divisions within the SQuaRE model are: ISO/IEC 2500n - Quality Management Division, ISO/IEC 2501n - Quality Model Division, ISO/IEC 2502n - Quality Measurement Division, ISO/IEC 2503n - Quality Requirements Division and ISO/IEC 2504n - Quality Evaluation Division. In particular, ISO/IEC 25000-Guide to SQuaRE represents the umbrella document of the SQuaRE series; it provides a general overview and guides to use the SQuaRE series. This document contains the SQuaRE architecture, terminology, intended users and associated parts of the series. ISO/IEC 25000 presents the whole SQuaRE series as a collection of quality engineering instruments. We are interested in the ISO/IEC 25030 (quality requirements), and the ISO/IEC 25010 (quality model, formerly called ISO/IEC 9126-1), which will be presented in what follows.

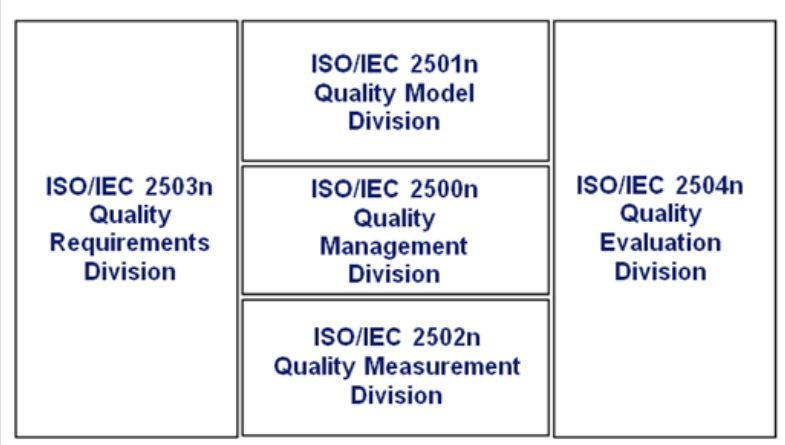


Figure 2 – Organization of the SQuaRE series of standards [21]

2.1 ISO/IEC 9126-1 and ISO/IEC 25010

Software products are constructed to be compliant with specific needs, required by its user. Their quality is determined in the measure that these needs are accomplished.

Software quality specification must be detailed and precise. The quality model is a way to formalize this specification [4]. In the SQuaRE family of standards, the ISO/IEC 25010 - Quality Model [22] will be an update of the current standard ISO/IEC 9126-1, with the same purpose of defining a quality model and providing a practical guidance on its use. According to this standard, a quality model is defined as a set of quality characteristics and their relations, providing a framework to specify quality requirements and to evaluate the quality of a software product, offering as well a common understanding and terminology of software quality. The six high level characteristics are refined into sub-characteristics, in a multiple levels hierarchy; the last abstraction level is constituted by the attributes, which are the measurable elements, to which metrics can be assigned. A quality characteristic of a software product is a set of properties describing and measuring this quality [23].

ISO/IEC 25010 is a revision of ISO/IEC 9126-1 [23], with minor changes. According to the draft version [22], it basically maintains the same definitions and structure of ISO/IEC (2001), however it offers eight characteristics: the same six characteristics of ISO/IEC (2001), plus interoperability and security, which were eliminated from the functionality sub-characteristics, for a total of eight high level characteristics. This choice responds to the quality requirements specification of current software applications, for example web services applications, where interoperability and security are architectural main concerns. This work will consider ISO/IEC 9126-1 because is the officially adopted standard.

The product quality is actually defined by three quality models: external, internal (see Figure 3) and in use quality model (see Figure 4). Internal quality refers to the static properties on the structure (such as the number of lines of code, modular complexity, number of faults found in a sequence or activity diagram) of the software product (conformed by all the intermediate products or artifacts) produced during the development process. It provides a "white box" view of the product. External quality is referred to software perspective on the computer system and it also refers to evaluate the software execution in a testing environment, on the computer hardware and applying an operating system (i.e. the measure of the number of faults detected during a test is related to the faults present in the program). It provides a "black box" view of product [23]. Finally the quality in use is perceived by the end users in their context, during the execution of the final software product. Internal quality has impact on external quality and this one on the quality in use. In this work, only the internal and external quality views are considered, since we are concerned about the early stages of the software development process.

2.2 ISO/IEC 25030

According to the recently adopted SQuaRE standard on quality requirements [21], the software system is usually part of a larger and more complex system; software requirements and system requirements are closely related and software requirements cannot be considered in isolation. In consequence it is important to consider the software quality requirements early in life cycle, as an important part of the specification of the software requirements. This standard focuses software quality requirements under a system perspective; they are categorized according to the quality model in the ISO/IEC 25010 standard (formally the ISO/IEC 9126-1 quality model standard). The quality model is hierarchically organized into characteristics and sub-characteristics, until the attributes, which are the measurable elements. The attributes specify the software quality requirements in terms of measures and target values. The standard

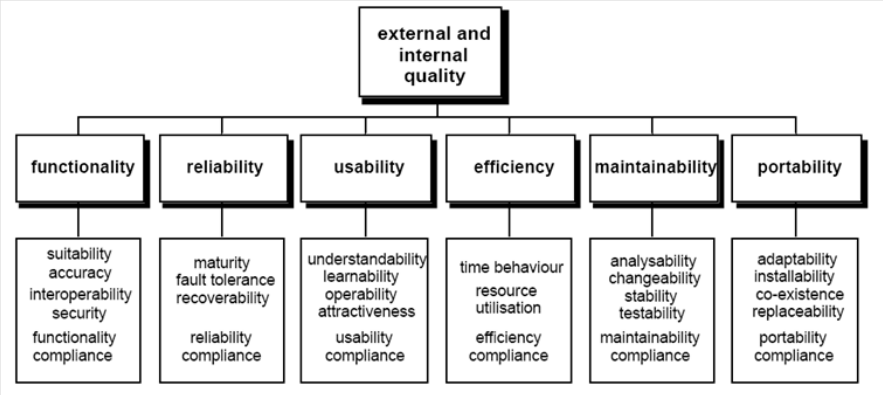


Figure 3 – External and Internal Quality Model ISO/IEC 9126-1 [23]

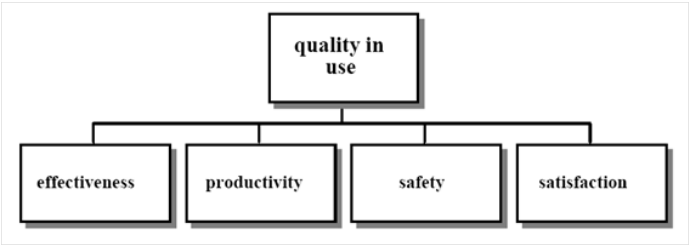


Figure 4 – In Use Quality Model [23]

provides recommendations and guidance to specify these requirements; it should help to ensure that they are in accordance with the stakeholders needs, stated clearly and precisely, correct, complete, consistent, verifiable and measurable. The ISO/IEC 25030 - Quality Requirements [21] standard helps specifying quality requirements either during software product quality requirements elicitation and analysis, or as an input for an evaluation process, and is a useful complement to ISO/IEC 9126-1. In particular, it provides a guide to identify software quality requirements, to validate the completeness of requirements specification and identify quality assurance and acceptance criteria for a software product. The system requirements are a precise formulation of stakeholders requirements, these requirements are considered as the technical view of requirements; system requirements are verifiable and state which characteristic the system has to possess in order to satisfy the stakeholders requirements. System requirements can include requirements for software, computer hardware, data, mechanical system, human business organization, etc., and may come from a variety of stakeholders including end users, organizations, and official bodies. ISO/IEC 25030 mainly focuses on software requirements (see Figure 5).

According to the categorization proposed in Figure 5, software requirements address either the software product or the software development process. Software product requirements include functional and quality requirements (inherent property requirements) and managerial requirements (assigned property requirements). Functional requirements include the application domain specific requirements as well as

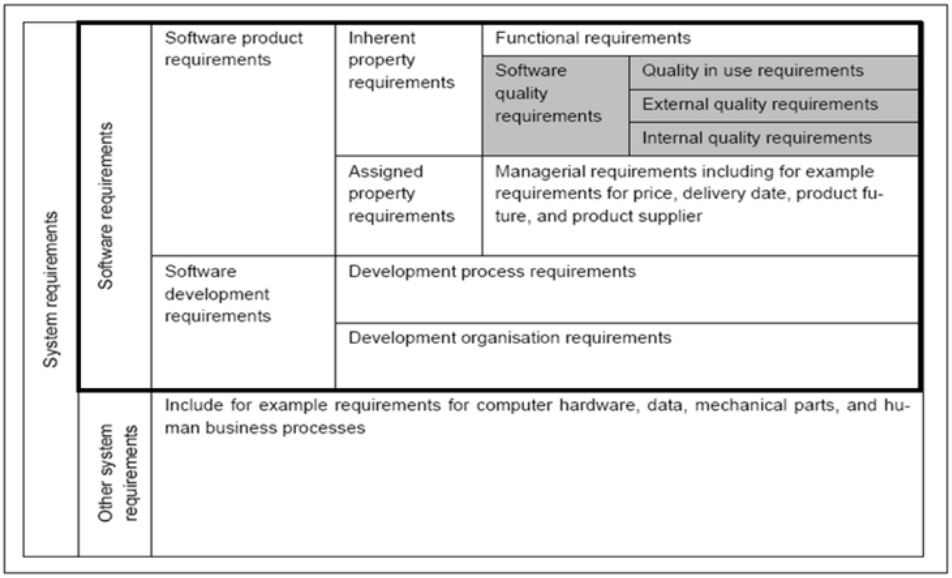


Figure 5 – System requirements categorization [21]

the user functional requirements; they require fulfilling quality goals, which are also a kind of quality requirements. Quality requirements may also imply architectural and structural requirements. Assigned property requirements represent requirements that can be changed without changing the software, including for example requirements for price, delivery date, product future and product supplier, so these are therefore not considered to be a quality characteristic of the software. According to ISO/IEC 25030, functional requirements determine what the software is able to do and quality requirements determine how well the software performs. In other words, quality requirements show the degree to which the software is able to provide and maintain its specified services. The software quality requirements have three different views (see section 2.1): quality in use, external quality and internal quality requirements.

In what follows, the conceptual model relating three worlds AOSD, Requirements Engineering and ISO/IEC Quality Standards is presented. It considers the AOSD terminology, the ISO/IEC 9126-1 (the official version, since ISO/IEC 25010 is still under discussion) for the quality properties specification, and integrates ISO/IEC 25030 with RECLAMO, for the software requirements categorization.

3 REASQ: Requirements, Aspects and Software Quality. A Conceptual Model

The REASQ conceptual model, expressed in UML [24], facilitates reasoning on the main notions inherent to an aspect-oriented quality requirements engineering discipline. In the model the software requirement, concern and quality characteristic elements are the main notions used to interrelate the terminology of three ambits: the requirements engineering discipline, the AOSD paradigm and the software product quality

specification [21, 23, 22, 35, 8]). They are denoted by REQUIREMENTS, ASPECTS and SOFTWARE QUALITY respectively in Figure 6.

3.1 Terminology

Basic concepts and terms used in the AOSD domain are extensions of the Aspect-Oriented Programming (AOP) [25, 26, 27] terminology, where they were used at first. In what follows the main terms are presented.

A concern is a property or interest point of a system [14, 17]. IEEE (2000) defines concerns as those interests related to the system development and operation, or any aspect that is critical and important for the stakeholder or participant in the software project. Concerns include system constraints, such as reliability, distribution, efficiency. In ISO/IEC 9126-1 [23] and ISO/IEC 25010 [22] a concern is a system requirement, i.e. a consideration that must be taken into account to satisfy a system goal. The concern notion appears in general to be related to a competence, an interest point or field. Kiczales et al. [25] defines crosscutting concerns are concerns that are found scattered in multiple modules or entangled in a unique module, such as data persistency, access control, transaction security, error handling. In [32], they are defined as those properties disseminated through the whole application code and that cannot be encapsulated within a functional unit or module, since they affect and crosscut the whole system. An aspect is defined in Filman et al. [14] as a modular unit designed to implement a concern, from an AOP point of view. They can contain code with the usual interface instructions: where, when, how to invoke it and what must be executed. In the ontology presented by Van den Berg et al. [37], an aspect is defined as a unit modularizing an otherwise crosscutting concern. In aspects languages, aspects are common modular implementation units defined by declarations of aspect type, similar to the class declaration in OO languages. Kiczales et al. [25] defines a program or code aspect as a modular unit appearing in other program's modular units; it is similar to a class. He also defines an aspect as the structure encapsulating a crosscutting concern [26, 27]. The scattering and tangling of code are troubles that the AOP and AOSD paradigm are trying to solve. AOSD arises from AOP and the need to identify and separate very clearly the crosscutting concerns, to produce a less entangled and hence more maintainable code, considering the whole software lifecycle [1]. The early separation of the crosscutting concerns and their composition into aspects are the main research trends in the so called early aspects discipline. In this paper an aspect is considered as a crosscutting concern, according to the definition present in the ontology of aspect orientation [37]. Separation of concerns (SOC) is an old term used first by Dijsktra [10] to enhance maintainability, as a principle to encapsulate characteristics in separate entities to locate possible sensitive points to changes and treat them separately in time. SOC has been defined as the ability to identify, encapsulate and manipulate software parts relevant to a particular purpose (i.e. concerns). SOC aims to reduce software complexity and improve understandability and traceability through a development process, minimizing the impact of changes during software evolution [19]. AOSD focuses on providing techniques to handle SOC. AORE [14] includes techniques to identify and model crosscutting concerns, introducing new mechanisms and abstractions to modularize and compose these concerns (i.e. identify match points, identify conflicts, identify the dominant concern, and define the composition rules). In Filman et al. [14], composition is defined abstractly as the idea of bringing together separately created software elements (i.e. an aspect involved with different core functionality modules). Non-functional requirements are

considered system constraints, such as usability and reliability, related with the quality of a service, i.e. the concern notion given by IEEE [20].

In the classic Requirements Engineering discipline, requirements are identified early in the lifecycle and are specifications of what the system shall do and how the system will be designed and implemented [35]. In Wiegers [39] they are descriptions of the system behavior and its properties. They can also be constraints on the development process. Wiegers [39] considers that software requirements include three distinct levels: business rules, user requirements and functional requirements. In addition, every system has an assortment of non-functional requirements. On the other hand, in Rosenhainer [33] a requirement is considered a special kind of concern. In general, a requirement specification is a textual description of the requirement expressed as a simple statement, representing exactly a concern, avoiding the mentioning of intercrossing with other requirements. However, it is generally accepted that some of these requirements appear or affect others and also the artifacts produced during the development process [14]. Generally, only two main groups of software requirements are considered: functional and non-functional requirements [35]. Functional requirements specify the software product functionality that must be implemented to satisfy the user tasks. Moreover they must satisfy the business goals and describe what the developer must implement. Non-functional requirements specify in general the constraints and/or rules imposed on the functionality of the product by the organization environment where it has to be executed.

3.2 The REASQ Model in details

It is assumed in general that requirements are descriptions of the system main behavior, of the properties or attributes of the system and/or of the constraints on the development process and/or the execution environment [35]. Software requirements are specified and derived from the system requirements, in the sense of ISO/IEC 25030. However we consider that the Business Rules, imposed by the organization requiring the software product, can impose some of the system requirements. Business rules can also come from an exterior organization that does not develop the software, like for example a governmental institution. They can be related to using existing components, with the life cycle, with the development methods and tools used (CMMI for example), with the operational environment (other platforms) or with interfaces in order to make the software more usable for other applications. Business rules are not themselves software requirements because they exist outside the boundaries of any specific software system, but they are inside the system boundaries. They often restrict who can perform certain use cases or they dictate that the software system must contain functionality to comply with the company policy. Sometimes business rules are the origin of specific quality goals that are implemented as a mechanism or component, in addition to the main user functionality. The softgoal model of Lamsweerde [28], the Non-Functional Framework (NFR) of Chung et al. [9], or the recent approaches of Brito and Moreira [5, 6], can be used to specify early this implicit functionality. Therefore, the origin of certain software requirements can be traced back to a particular business rule [39]. According to general agreement [35, 39, 8], in the literature we found that software requirements are classified into functional and non-functional requirements; in ISO/IEC 25030 non-functional requirements correspond directly to the quality requirements (see Figure 5). So in the model we have the inheritance relation to express that functional and quality requirements are a specialization of the inherent property requirements (see Figure 6). Functional requirements capture the

system behavior and can be expressed as services or components that must also fulfill precise quality goals. Quality requirements specify the conditions that the system must satisfy to constrain, condition or control the execution of the system components.

According to ISO/IEC 25030, a quality requirement is associated to a quality characteristic and it is defined by a quality model (ISO/IEC 25010), which is used to characterize the application domain. Quality requirements according to ISO/IEC 25030 are categorized by software quality in use, external and internal requirements (expressed by inheritance relations). This specialization represents the software quality view of requirements. In Figure 6 we have left the tag "ISO/IEC 25010 Quality Model" to maintain the new SQuaRE family standards, but must be followed the guidance proposed by ISO/IEC 9126-1. Business rules impose some of the software requirements and originate implicit functionality. An implicit functionality can be considered as a functional requirement not directly expressed by the user. In ([8]), it is focused as a component introduced to respond to a specific quality requirement or to a business rule, in the same sense as the operationalization feature in the NFR framework (Chung et al., 2000). Moreover, when quality requirements are refined, they are sometimes expressed as functional requirements (implicit functionality). On this basis, quality requirements are specialized into global and specific quality requirements (expressed by inheritance relation in the model), showing the requirements view. This approach can be traced back to the early works of Dromey [11]. A global quality requirement is considered ascribed to the overall system configuration, such as performance, reliability, portability and interoperability, meaning that they must be considered by every component in the configuration (see Figure 6). Global quality requirements however, are often implemented as architectural styles, or by process mean (review, testing, etc.). They are in particular related to the system overall architectural style or context, like for example a SOA (Service Oriented Architecture) and the network environment. A global quality requirement may only be relevant for a subset of the software, a set of functions or subsystem. On the other hand, a specific quality requirement such as security or accuracy can be satisfied by adding individual components, as an implicit functionality, implying a mechanism to "operationalize" the requirement. They can be proposed as "candidate aspects", since they are like to crosscut functionalities.

The establishment of this quality requirements taxonomy is important to reduce complexity in the overall requirements specification of the application. Quality characteristics are associated to inherent property requirement, for example the "interoperability" quality characteristic is associated to the "communication among heterogeneous platforms is needed" quality requirement. With respect to the ASPECT ambit, a concern defines a software requirement and it is specialized into functional, non-functional and crosscutting concerns. Notice that functional and non-functional concerns could be omitted in the model, for the relation existing with the inherent property requirements, which is straightforward. We have left them here to show that a concern can crosscut both functional and non-functional requirements. Moreover a crosscutting concern is implemented by an aspect and so an aspect can also implement an implicit functionality (provided it crosscuts some concern). A composition mechanism brings together separately created software elements (i.e. aspects with core functionality modules), an aspect is defined by a composition mechanism which has associated a pointcut. A pointcut is an expression that identifies a specific point or a set of points in the execution of a system. The joint points where the associated aspect advice should be executed, is also defined by the composition mechanism. A joint point is a point of interest in some artifact in the software lifecycle through

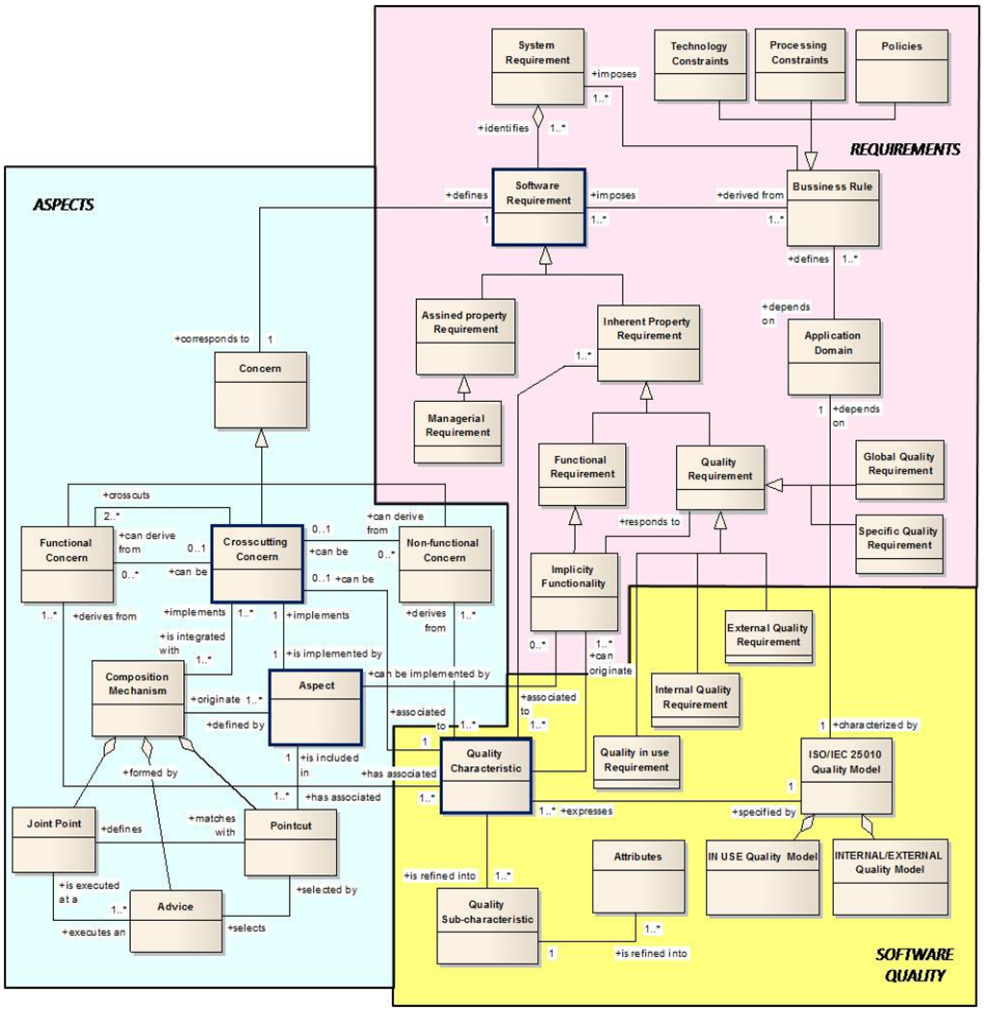


Figure 6 – REquirements, Aspects and Software Quality Model

which two o more concerns may be composed. An advice provides the actions that will occur at the joint points and augments or constraints other concerns at the joint points, matched by a pointcut. Pointcut, joint point and advice are essential elements to implement an aspect [27, 37]. A quality characteristic, derived from an inherent property requirement or from an implicit functionality, can be a potential crosscutting concern. This issue is very important for the treatment of aspects at early stages of software development, where the notion of candidate aspect is introduced [6]. In this context, a crosscutting concern can be functional or non-functional (represented by the specialization relation in the model in Figure 6). The ISO/IEC 25030 framework is concerned only with the REQUIREMENTS ambit of the general system and does not deal with the ASPECTS ambit; in the REASQ model instead, the relations between these two ambits is established. Moreover, the software quality ambit of ISO/IEC 25010 (formally ISO/IEC 9126-1 must be used) is also modeled and related with the other two. To conclude about the discussed terminology involved in three important

software engineering research topics, a concern corresponds to a software requirement [33] that must be handled to solve some software problem [27]; it can be used in the requirements engineering discipline and in architecture modeling. Using REASQ, a mapping is established between the ISO/IEC standards and the emerging AOSD discipline. Actually, non-functional concerns and quality requirements are related with one or more quality characteristics of the standard quality model (potential crosscutting concerns), which is a main goal of AOSD [1, 14]). There is a general agreement on the fact that an aspect solves the problem of the crosscutting concerns (provided these are identified), by encapsulating them in a modular structure, through a composition mechanism [1, 12, 14]. This can be done early in the software development process through a composition table [6], while modeling the system architecture, for example, to facilitate the design and implementation stages.

3.3 Case Study: Instantiating the REASQ Model

This case study is based on a simplified version of an online toy store (an e-commerce web-based application). A customer (web user) can buy a toy online registering an account and specifying a login. The system provides options to the customer to browse through the toy store on-line catalogue. A customer can add the currently selected toy to, or remove from, his shopping car. When the customer wants to checkout, his goal is to buy the current contents of their shopping car (using credit card), including paying for the content and arranging delivery. Additionally, a customer can check the order status and cancel an order (only if it has not been processed). A customer demands to the system is secure operations, fast processing, and easy browsing through the catalogue. The system must operate 365 days of the year.

In what follows, the most important elements of the REASQ model are illustrated with the Toy Store Case Study. Figure 7 presents a simplified version of the use case model, where we can identify the most important functional (use cases) and specific quality requirements (included use cases as candidate aspects).

3.3.1 Business Rules

Some business rules to the Toy Store Systems are presented as follows.

- A customer must create an account to buy a toy online.
- A customer must have access from an arbitrary web browser.
- Only payments with credit card must be accepted.
- Credit card approval time must be less than 2 minutes.

3.3.2 Functional Requirements (from Inherent Property Requirements) and Quality Requirements

An e-commerce web-based application is a portal type application that uses mainly transactional and portal services. For this kind of applications, general concerns (global quality requirements) are identified. The main overall functional requirements and the architectural style are known studying the application domain [23]. They are presented in what follows.

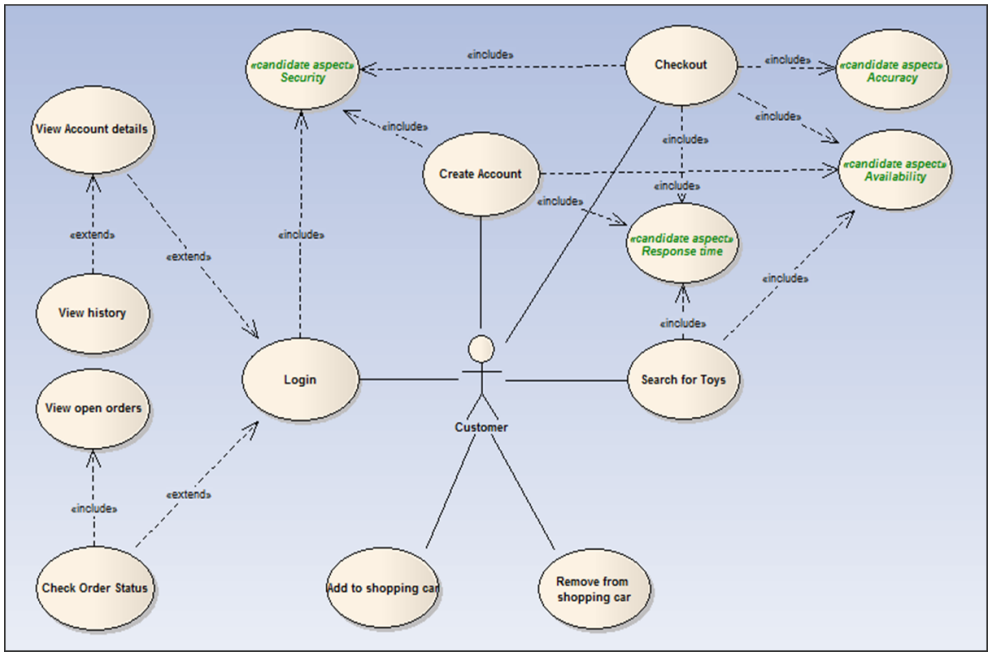


Figure 7 – The Toy Store Use Case Model

Global Quality Requirements

Transactional

- Functionality: security (integrity), accuracy.
- Reliability: availability
- Efficiency: response time, resource utilization

Portal

- Portability: adaptability (scalability)
- Efficiency: response time, resource utilization

Specific Quality Requirements

From the problem description we can identify the main functional concerns and then the specific quality requirements for the Toy Store System: login (security, efficiency), create account (availability, efficiency), search for toys (accuracy, efficiency, availability), checkout (security, efficiency, interoperability). In table 1, and following the early aspect approach [5, 6, 12] these specific quality requirements have been including as candidate aspects (potential crosscutting concerns). Non-functional concerns (quality requirements) are identified and related with the functional requirements to identify relevant crosscutting concerns. Table 1 presents some relevant REASQ elements and synthesizes the composition of the main functional and quality requirements, focusing on the relevant crosscutting concerns, denoted as "candidate aspects". The crosscutting concerns are actually quality goals that the functional concerns must accomplish and can be implemented as aspects. Figure 7 illustrates the crosscutting concerns, represented as stereotypes called "candidate aspects" use cases. Note that accuracy appears in only one functional requirement; hence it is not a crosscutting concern.

Functional Requirement	Quality Goal	Crosscutting Concern («candidate aspect »)
Login	- Secure access and user validation must be provided.	- security
	- A friendly interface must be provided.	- usability
Create account	- Communication between other systems must be guaranteed.	- interoperability
	- A friendly interface must be provided.	- usability
	- Secure access and user validation must be provided	
	- security	
Search for toys	- Communication between catalogues.	- interoperability, availability
	- Fast responses must be guaranteed.	- response time, usability
	- A friendly interface must be provided.	
Checkout	- Fast communication between other systems must be guaranteed.	- response time, interoperability
	- Reliable and secure payment operations are required.	- security, accuracy
	- A friendly interface must be provided.	- usability

Table 1 – Composition: Functional Requirements, specific quality requirements and identified concerns

Interoperability and availability depends from the networking environment and the service availability in a Service Oriented Architecture (SOA) style, and finally the user interface component must deal with usability.

Quality Model

In particular, in our online Toy Store System the quality model has been defined from the global and specific requirements shown above and is summarized as:

- Functionality: suitability, accuracy, security and interoperability.
- Reliability: maturity, fault tolerance (availability).
- Usability: understandability, operability, attractiveness.
- Efficiency: time behavior, resource utilization.
- Portability: adaptability.

4 Applying the REASQ Model to Define an Ontology

In what follows, the REASQ model will be used to define an ontology conformed by three related ontologies corresponding to each ambit. In this way, the semantics

expressed by the model will be formalized and can be easily reused by the software community. An ontology is a formal and explicit specification of a conceptualization [16], and contains the description of the concepts and relationships which exist in some domain, using a formal language. It conforms a shared vocabulary, where domain rules are also important. An ontology can be viewed as a shared knowledge of a domain that can be communicated between humans and computer systems. Ontologies can be integrated, reused and shared by different application domains, explaining the growing interest of the scientific community in this area [15]. Several works have been developed about ontologies involving software quality. Particularly, in Abran et al. [2] the authors present an approach to build a Software Engineering Body of Knowledge (SWEBOK) ontology to increase internal consistency and clarity. A specific example of the benefits of an ontology is presented, along with an analysis of the term "quality" in the current version of the SWEBOK Guide. The ontological approach has been proposed for the WS (Web Service) domain mainly to unify notions on QoS (Quality of Service). On the other hand, in W3C [40] an ontology and language have been defined to improve the search of WS; in the WSMO language, WS descriptions consist of non-functional, functional, and the behavioral aspects of a Web service. In Losavio et al. [30] a view of the WS domain knowledge based on software quality is presented, using an ontology to formalize its characterization and information retrieval. This ontology integrates three standards on software product quality at different abstraction levels, to unify terminology and characterize reusable domain knowledge; on the other hand, to facilitate web services identification based on their quality properties.

4.1 Domain and scope of the REASQ Ontology

This section presents three related ontologies written in Protégé [18] and derived from the REASQ model. These ontologies will cover the three REASQ ambits: aspect-orientation, requirements engineering and software quality and will be integrated in a general ontology named REASQ Ontology which will be used for the following purposes:

- To integrate the main concepts used in AOSD with related requirements engineering notions, and the recent ISO/IEC 25030 software quality requirements categorization.
- To set the basis for a better understanding and consensus towards a common and standard vocabulary for the emerging aspect oriented requirements engineering discipline.
- To favor domain knowledge reutilization by the software engineering community.
- To present an umbrella for the definition of an aspect-oriented quality requirements engineering process.

The information in the ontology should provide answers to the following main questions:

- How aspect-orientation terminology is related with requirements engineering terminology?
- How aspect-orientation terminology is related with software quality terminology?

- How the requirements engineering terminology is related with the software quality terminology?
- How concerns are defined and related with software requirements?
- How crosscutting concerns are related with quality requirements?
- How software requirements are categorized according to the ISO/IEC 25030 standard?
- What are the quality characteristics of a quality model for a given application?

The structure of the REASQ model has been formalized with the specification of the three ontologies mentioned above and shown in Figure 8. Moreover, Figure 9 presents the three ambits related ontologies as the REASQ Ontology, which shows the main notions of the model, to facilitate understanding and reuse. They have been implemented with the Protégé 3.4 version [18, 40]. The Protégé notation considers rectangles to represent concepts (classes); relations among concepts are represented by dotted lines with annotated roles. A full line indicates a hierarchy of concepts (class generalization) with the "is-a" role (see Figure 8).

4.2 Terms

The list of the terms used in the ontologies corresponds to the terminology integrated by the REASQ conceptual model (see section 3.2).

5 Related Work

There are few related works linking the aspect-oriented approach, the ISO/IEC 9126-1 standard to specify quality requirements and the discipline of requirements engineering. We greatly favor the approaches using standards, because we believe that their use is needed for a mature Software Engineering discipline, facilitating understanding and communication among stakeholders. In Navarro et al. [31] is presented a proposal to organize software requirements by integrating aspect-oriented techniques within a goal-oriented approach for requirements. In their work, ISO/IEC 9126-1 is used as a starting point to establish the possible concerns, relating the requirements with the ISO/IEC 9126-1 quality characteristics. In particular, in this work, ISO/IEC 9126-1 provides an initial framework to elicit and organize goals and requirements and then provide a wide set of concerns. A methodology for the concurrent definition of software architectures is also used (ATRIUM). This model provides an initial view of the concerns that could be meaningful for the system. They indicate that the analyst can iteratively select what he/she considers proper and initiate its specification and refinement, as they elicit the requirements; it could be convenient to incorporate new concerns to properly include additional goals/requirements. Comparing with REASQ model, ISO/IEC 9126-1 (now called ISO/IEC 25010) is used to relate the requirement engineering terminology (following the new standard ISO/IEC 25030) with the aspect-orientation and software quality, where a quantifiable quality characteristic is directly associated with a crosscutting concern.

On the other hand, in Schauerhuber et al. [34] an initial version of a reference architecture for Aspect-Oriented Modeling (AOM) is presented; this AOM reference architecture represents a set of general assertions and normative assertions, which are

true for AOM languages, as well as a technique for designing new AOM languages or for extending existing (domain-specific) modeling languages with concepts of the aspect-oriented paradigm. The reference architecture is presented as a UML class diagram; the concepts used in the reference architecture have been refined to make them suitable for the modeling level. In this work the concepts point out the design level view, constituting a proper basis for a later code generation step, because reengineering code at a certain point within a program requires the consideration of the full spectrum of modeling concepts not present in programming languages. In the REASQ model the terminology points out the establishment of a unique conceptual model to clarify the AOSD emergent terminology from the requirements level view, and it focuses more its usage at the early stages of software development, for example in architecture modeling. Van den Berg et al. [37] present a report about a first public version of ontology of aspect orientation, in particular a glossary for common AOSD terms, and a proposal for a conceptual framework for crosscutting (which could be part of the taxonomy for AOSD) are presented. They describe the definitions of an initial set of core terms in AOSD. The initial definitions are based on the AOSD book by Filman et al. [14]. After several reviews, they selected preferred definitions that are more general, and for which there is general agreement. The REASQ Model, also presents a taxonomy with concepts and their relations about the most representative terms of aspect orientation (conceptual domain model), but these have been interrelated with the new system requirements categorization of ISO/IEC 25030 [22] and the software quality ambit of ISO/IEC 25010. The quantification of the quality characteristics related to the concerns can help in the assignment of priority to a particular concern or to solve tradeoffs among conflicting concerns, according to a soft-goal approach, for example. Finally in Vaníček [38] the standard series of SQuaRE is presented briefly, as a contribution that tries to define the stakeholders requirements for software products, to facilitate objective quality evaluation.

6 Conclusion

This work presents a common framework, a UML conceptual model called the REASQ model and its specification as an integrated ontology implemented with the Protégé tool, for the reasoning, understanding, handling and reuse of the main notions related to the aspect oriented software development paradigm. REASQ integrates three main research ambits in software engineering: classical requirements engineering, standard software quality requirements and emerging AOSD. We believe that the use of standards, even if it may seem restrictive, is in general a step towards the common understanding of interacting working groups of different interests and nature, which is the case of the software project development team and a main cause of software failures. An example of the standards adoption by the software community is UML. The REASQ model and ontology is a useful tool at early stages of development, for example during the modeling of the system architecture, when requirements must be clearly identified, classified and quantified. In the Web Services domain for example, the ontology can ease the retrieval of the right metrics corresponding to a particular quality of service, facilitating the search for the appropriate service. REASQ has been used in different software engineering post graduated course projects and in some research projects. We are working to use it to support the definition of a quality requirements engineering process. In particular, the REASQ model and the derived ontology is being applied to the characterization of the dependable systems domain,

where the early handling of crosscutting concerns is of major importance.

Acknowledgments This work has been partially supported by the Consejo de Desarrollo Científico y Humanístico (CDCH) of the Central University of Venezuela, ADIRE project, PG-03-7310-2008/1 and the OPSU (Oficina de Planificación del Sector Universitario).

References

[1] AOSD, 2008. Aspect-Oriented Software Development Home Page. Available from: http://www.aosd.net/wiki/index.php?title=Glossary

[2] Abran, A., Moore, J., Bourque, P., Dupuis, R., Tripp, L., 2003. Guide to the Software Engineering Body of Knowledge - SWEBOK, Trial Version 1.0. IEEE-Computer Society Press. Available from: http://www.swebok.org

[3] Bass, L., Klein, M., Bachmann, F., 2001. Quality Attribute Design Primitives and Attribute Driven Design Method. 4th International Workshop on Product Family Engineering. Bilbao, Spain, 3-5.

[4] Bertoa, M., Troya, J., Valecillo, A., 2002. Aspectos de Calidad en el Desarrollo de Software Basado en Componentes. Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga.

[5] Brito, I., Moreira, A., 2003a. Advanced Separation of Concerns for Requirements Engineering. VIII Jornadas de Ingeniería del Software y Base de Datos, JISBD 2003. Alicante, Spain. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.6438&rep=rep1&type=pdf

[6] Brito, I., Moreira, A., 2003b. Towards a Composition Process for Aspect-Oriented Requirements. Early Aspects Workshop at AOSD Conference. Boston. Available from: http://www.cs.bilkent.edu.tr/AOSD-EarlyAspects/Papers/BritoMoreira.pdf

[7] Boegh, J., 2008. A New Standard for Quality Requirements. IEEE Software, vol. 25, No. 2, March/April, 57-63.

[8] Chirinos, L., Losavio, F., Matteo, A., 2004. Identifying Quality-based Requirements. Information Systems Management (ISYM). Auerbach Publications, 21(1), 15-21.

[9] Chung, L., Nixon, B., Yu, E., Mylopoulos, J., 2000. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers.

[10] Dijsktra, E., 1976. A Discipline of Programming. Englewood Cliffs, NJ. Prentice Hall.

[11] Dromey, R., 1996. Cornering the Chimera. IEEE Software, vol. 13, No. 1, 33-34.

[12] Early-Aspects, 2008. Aspect-Oriented Requirements Engineering and Architecture Design Home Page. Available from: http://www.early-aspects.net

[13] Falbo, R., Guizzardi, G., Duarte, K., Natali, A.C. 2002. Developing Software for and with Reuse: An Ontological Approach. Conference on Computer Science, Software Engineering, Information Technology, e-business and Applications. Ischia, Italy, pp477-488.

[14] Filman, R., Elrad, T., Clarke, S., Aksit, M., 2005. Aspect-Oriented Software Development. Addison Wesley, Boston.

[15] Gómez, A., Corcho, O., Fernández, M., 2003. Ontological Engineering. Advanced information and knowledge processing Series. Berlin: Springer Verlag.

[16] Gruber, T. R., 1993. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220.

[17] Gutiérrez, J., Villadiego, D., Escalona, M., Mejías, M., 2004. Aplicación de la Programación Orientada a Aspectos en el Diseño e Implementación de Pruebas Funcionales. DSOA'2004, IX Jornadas de Ingeniería de Software y Bases de Datos. Málaga.

[18] Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C., 2004. A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools, Edition 1.0. University of Manchester.

[19] IBM-Research, 2000. Workshop on Multi-Dimensional Separation of Concerns, International Conference on Software Engineering, ICSE'2000. Available from: `http://www.research.ibm.com/hyperspace/workshops/icse2000`

[20] IEEE, 2000. Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471-2000.

[21] ISO/IEC 25030, 2006. Software Engineering. Software Product Quality Requirements and Evaluation (SQuaRE). Quality Requirements.

[22] ISO/IEC CD 25010, 2007. Software Engineering. Software Product Quality Requirements and Evaluation (SQuaRE). Quality Model and guide.

[23] ISO/IEC 9126-1, 2001. Information Technology - Software Engineering Product Quality. Part 1: Quality Model.

[24] Jacobson, I., Booch, G., Rumbaugh, J., 2000. El Lenguaje de Modelado Unificado. Segunda Edición. Madrid: Addison Wesley.

[25] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J., 1997. Aspect-Oriented Programming. ECCOP'97 Object-Oriented Programming, 11th European Conference, M. Aksit y S. matsouka, Eds. LNCS 1241, 220-242.

[26] Kiczales, G., Lieberheer, K., Ossher H., 2001. Discussing Aspects of AOP. Communicatios of the ACM. Vol.44 No.10, 33-38.

[27] Laddad, R., 2003. AspectJ IN ACTION. Practical Aspect-Oriented Programming. Manning Publications.

[28] Lamsweerde, A., 2003. From system goals to software architecture. In M. Bernardo and P. Inverardi, editors, SFM, Formal Methods for Software Architectures. LNCS Springer-Verlag, 25-43.

[29] Losavio, F., Chirinos, L., Levy, N., Randane-Cherif, A., 2003. Quality Characteristics for Software Architecture. Journal of Object Technology, 2(2), 133-150.

[30] Losavio, F., Matteo, A., Levy, N., 2009. Web Services Domain Knowledge with an Ontology on Software Quality Standards. 3rd International Conference on Internet Technologies and Applications (ITA'09), CAIR (Centre for Applied Internet Research) Glyndwr University, 8-11 September, Wrexham, U.K, 74-85.

[31] Navarro, E., Letelier, P., Ramos, I., 2004. Goals and Quality Characteristics: Separating Concerns. In Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (AOSD). Lancaster.

[32] Reina, A., Torres, J., Toro, M., Álvarez, J., Nieto, J., 2003. Una experiencia Práctica reutilizando Aspectos. Actas del Taller de Trabajo de Desarrollo de Software Orientado a Aspectos DSOA'03. Alicante.

[33] Rosenhainer, L., 2004. Identifying Crosscutting Concerns in Requirements Specifications. In Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (AOSD). Lancaster.

[34] Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M., Kappel, G., 2006. A Survey on Aspect-Oriented Modeling Approaches. 2006.

[35] Sommerville, I., Sawyer, P., 1997. Requirements Engineering. A Good Practice Guide. John Wiley and Sons, New York.

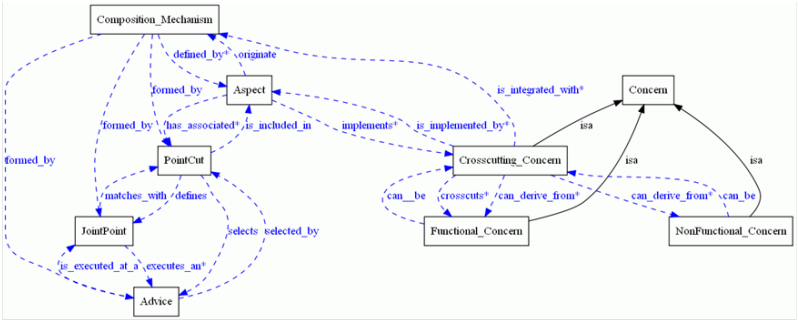
[36] Stefani, A., Xenos, M., 2001. A model for assessing the qualityof e-commerce systems. Proceedings of the PC-HCI 2001 Conference on Human Computer Interaction, Patras, 105-109.

[37] Van den Berg, K., Conejero, J., Chitchyan, R., 2005. AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation. Technical Report AOSD-Europe-UT-01, AOSD-Europe. Available from: `http://www.comp.lancs.ac.uk/computing/aosd-europe//deliverables/d9.pdf`

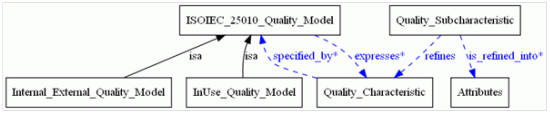
[38] Vaníček, J., 2006. Software Quality Requirements. Agric. Econ. –Czech, 52,177-185. Available from: `http://www.cazv.cz/attachments/ZE_52_177-185.pdf`

[39] Wiegers, K., 2003. Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Microsoft Press, Washington, USA. Available from: `ftp://ftp.ifs.uni-linz.ac.at/pub/publications/2006/1406.pdf`

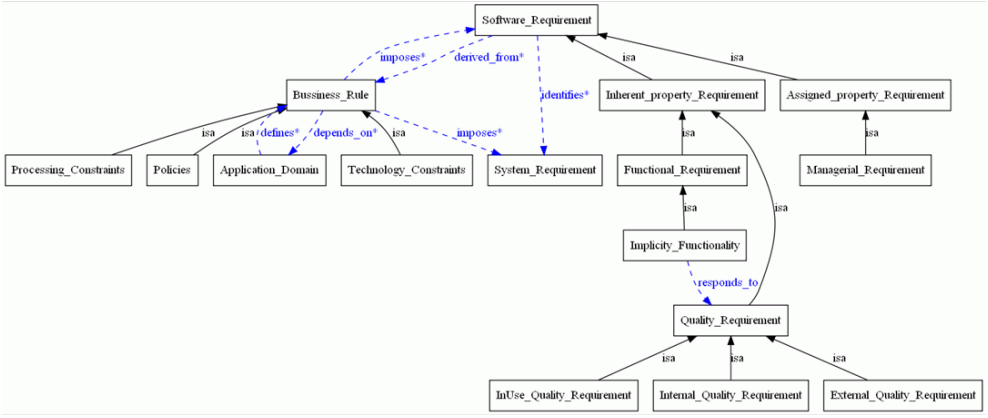
[40] W3C, 2005. World Wide Web Consortium. Web Service Modeling Ontology. W3C Member Submission 3 June 2005. W3C Working Draft 28 October (2002). Copyright 2005 W3C.



(a)



(b)



(c)

Figure 8 – Conceptual Models for: (a) Ontology of Aspect-Orientation ambit, (b) Ontology of Software Quality ambit, (c) Ontology of Software Requirements ambit

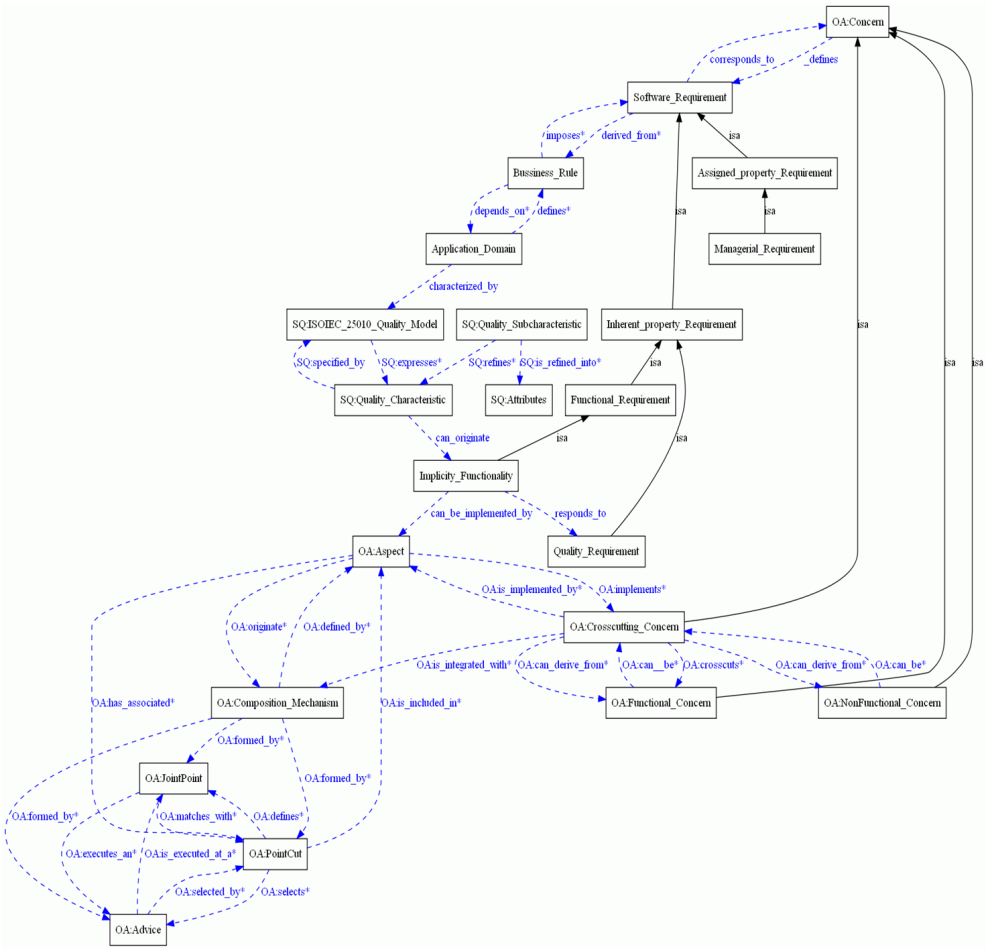


Figure 9 – Integrated Conceptual Model: The REASQ Ontology relating Aspect-Orientation, Software Requirements and Software Quality.

About the authors

Isi Castillo is Assistant Professor in the Department of Computing at the National Experimental University of Sur del Lago "Jesús María Semprúm", Santa Bárbara de Zulia, Venezuela. She is currently a Doctor Candidate in Computer Science at the Central University of Venezuela, Caracas, Venezuela. Additionaly, she works at the MoST (Models, Software & Technology) Laboratory of the ISYS (Software Engineering and Systems) Research Center and her research interests focus on the Software Engineering field, and specifically on aspect-oriented software development, aspect-oriented requirements engineering and quality software. E-mail: `castilloi75@gmail.com`

Francisca Losavio received the Doctor degree in 1991 and a 3ème. Cycle Doctor Degree in 1985, both in Computer Science and from the Paris-Sud University, Paris XI, Orsay, France. She also obtained a MSc. Degree also in Computer Science in 1983 from the Simon Bolivar University, Venezuela. She is a Titular Professor at the School of Computer Science, Faculty of Science, Venezuela Central University, Caracas, where she works at the ISYS (Software Engineering and Systems) Research Center, coordinating the MoST (Models, Software & Technology) Laboratory. She has participated in national and European Community research projects. Her main research axes are software architecture, software quality, quality standards and software development process. E-mail: `francislosavio@gmail.com`

Alfredo Matteo received the Doctor degree in Computer Science for the Paul Sabatier University, Toulouse, France in 1984. At present he is Titular Professor at the School of Computer Science, Faculty of Science, Venezuela Central University, where he has coordinated the TOOLS Laboratory of the ISYS (Software Engineering and Systems) Research Center, being now part of the research staff of the MoST (Models, Software & Technology) Laboratory of ISYS. He is now responsible of the Postgraduated Studies in Computer Science. His research includes software engineering, requirements engineering, architectures, methodologies and model driven engineering. E-mail: `alfredo.matteo@ciens.ucv.ve`

Jorgen Boegh is the Safety and Quality Manager at Terma S/A. He is head of the Danish Delegation to ISO/IEC JTC1/SC7 and is editor of three international standards in the area of software quality requirements and evaluation. His research interests include software quality modeling, requirements specification, and quality evaluation. He received his MSc in mathematics and computer science from the University of Aarhus. E-mail: `jbh@terma.com`