Non-Functional Requirements
as drivers of
Software Architecture Design

David Ameller

Thesis supervised by

Dr. Xavier Franch

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computing

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Abstract

In the last decades, software engineering has become an important area of research. As researchers, we try to identify a problem, a need, or a hole in some research topic, once identified we make an effort to produce new techniques, methods, and tools that hopefully will help to improve the detected issue. In the present thesis the identified issue was the need of *supporting non-functional requirements in the software architecture design* where these requirements are the drivers of the architectural decision-making.

This thesis started with the idea that a relatively new software engineering discipline, model-driven development, was a good place to propose a solution for the detected issue. We envisioned how non-functional requirements can be integrated in model-driven development and how this integration will impact in the architectural design activities.

When we started to produce our techniques, methods, and tools for model-driven development we found out that there was a bigger hole in the web of knowledge than what we had initially foreseen. Much of the evidence of how non-functional requirements affect the software architecture design is hidden. This situation caused a turn in this thesis: we needed to understand architects, how they think and how they make the architectural decisions, what is the role of non-functional requirements in the architectural decision-making process, and to what extent are the non-functional requirements important in this process. All these questions needed an answer, an answer that only architects could provide. In consequence we opted to drove several empirical studies to answer these questions.

In parallel, we started to work in a way of representing this knowledge, an ontology for software architecture that integrates non-functional requirements. Using this ontology as basis, we designed a method to assist architects in the architectural decision-making process and a tool that acted as a proof of concept of both, the ontology and the method.

In summary, this thesis explores how non-functional requirements are currently integrated in the software architecture design practices, and proposes ways to improve this integration and facilitate the work of architects by providing means to assist them in the architectural decision-making process.

Co-authorship statement

Most of the contents in this thesis are based on published papers authored by the candidate, and in some cases co-authored with other authors. The contents of the papers included in this thesis may have been adapted, reorganized, and extended with respect to the published version. The contribution for each chapter is described, including specifically who contributed to the work and the nature and extent of his/her contribution.

All thesis chapters

Dr. Xavier Franch, as supervisor, has contributed to the research described in this thesis. Xavier's has been active and involved in the research being conducted, discussing, and writing each paper. His guidance and supervision has been fundamental for the selection of conference and journal targets for publications and the improvement of English writing skills.

Chapter 3

This chapter is based in [24], which was co-authored with Dr. Jordi Cabot, associate professor at the École des Mines de Nantes and the leader of the AtlanMod team. Jordi participated in the discussions of this research giving his opinion, in particular his expertise in Model-Driven Development was of great value for this research. He also helped in the writing of the paper.

Chapter 5

This chapter is composed by three empirical studies. While the first one was performed with Dr. Xavier Franch alone, the other two were in collaboration with other researchers. The second empirical study is based in [11, 12], which was co-authored with Dr. Claudia Ayala, researcher at the Universitat Politècnica de Catalunya and Dr. Jordi Cabot, previously introduced. Claudia was the expert in driving empirical studies, and proposed the protocol to follow, she also participated in the execution of the study and in the writing of the papers. Jordi participated in most of the discussions of this research and helped in the writing of the papers. The third empirical study is based in [25], which was co-authored with Dr. Matthias Galster, researcher at the University of Canterbury, and Dr. Paris Avgeriou, researcher at the University of Groningen and leader of the SEARCH group. Matthias produced the first version of the research protocol and helped in the writing of the paper. Paris participated in most of the discussions of this research giving his opinion, in particular his expertise in software architecture was of great value for this research. He also helped in the writing of the paper.

Chapter 7

The Section 7.3 of this chapter is based in the papers [14, 15], which were co-authored with Oriol Collell. Oriol was the main developer of the ArchiTech tool, and he also helped in the preparation of the tool demo and the promotional video.

Acknowledgments

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his supervisor, who was abundantly helpful and provided extremely useful assistance, support, and guidance.

Special thanks also to all his research group members and university colleagues; for their invaluable assistance. Not forgetting his best friends who always have been there.

The author wishes to express an special gratitude to his mother and father, and to his relatives; for their support during his studies.

The following is the list of people, in alphabetical order, that helped in one way or another to the completion of this thesis.

Antonio Vallecillo
Antonio Villegas
Borja Balle
Carles Farré
Carlos Ameller
Charlie Ameller
Claudia P. Ayala
Cristina Gómez
Cristina Palomares
David Aguilera
David Cerdan
David Ruiz
Frank Buschmann
Hugo H. Pibernat
Jaelson Castro
Jordi Cabot
Jordi Marco
Judith Mitchell
Lidia López
Marc Oriol
Matthias Galster
Nadia Ameller
Oriol Collell
Oscar Cabrera
Oscar Hernan
Oscar Pastor
Paris Avgeriou
Paul Grünbacher
Raul Marina
Rebeca Dalmau
Silverio Martínez
Vicente Pelechano
Xavier Franch

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and terminology

The present thesis has grown around three software engineering areas: Requirements Engineering (RE), Software Architecture, and Model-Driven Engineering (MDE). The three areas are explained in the following subsections to clarify the context and the terminology used in this thesis.

1.1.1 Requirements Engineering

Requirements engineering is the part of software engineering which covers all of the activities involved in eliciting, documenting and maintaining requirements [180]. This thesis does not deal with the elicitation of requirements since it is assumed that the requirements are already defined. Some of the contributions may help the documentation and the maintainability of requirements. The principal contribution with regard to software requirements is to make them present and drivers of the subsequent development activities.

There are several classifications of requirements, the most habitual one being the differentiation between functionality and non-functionality. For example, a functional requirement could be: *the system shall produce an inventory every week*; while a non-functional requirements could be: *the system shall load web pages in less than 2 seconds.* A second classification, is the differentiation between technical and non-technical requirements.

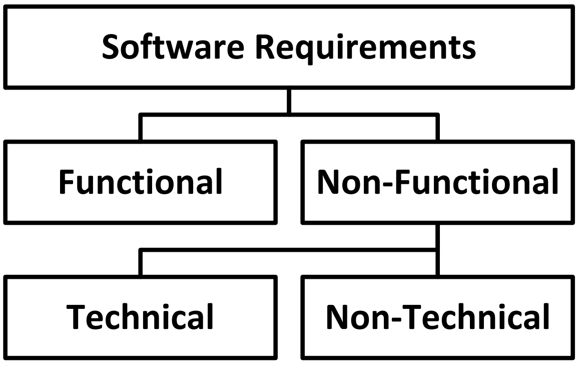


Figure 1.1: Classifications of software requirements

Functional requirements are by definition technical, but non-functional requirements could be inherent to the software being produced (technical), or could be triggered by external factors such as organizational, laws, software licensing, software providers, in these cases the requirements are classified as non-technical requirements (e.g., *the system shall be developed in a language known by the development team*). Figure 1.1 illustrates the explained classification.

Non-Functional Requirements

Non-Functional Requirements (NFRs) are one of the main research targets in the Requirements Engineering community [98] and their impact has been documented in seminal papers [48], individual case studies [90] and types of industrial projects [215]. For the present thesis, NFRs are of special relevance.

There are many NFR definitions (see [62, 98]), one definition is:

> *"[NFR is...] a requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. [NFR is...] a requirement that specifies physical constraints on a functional requirement"*, I. Jacobson et al., "Unified Software Development Process", 1999 [120].

L. Chung et al. [62, 63] discussed that the lack of integration of NFRs with functional requirements can result in long time-to-market and more expensive projects. This fact has been recurrently mentioned in previous publications such as the book “Software Requirements: Objects, functions and states”, A. M. Davis [75]. This is the first paragraph of [62]:

> *“Essentially a system’s utility is determined by both its functionality and its nonfunctional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the system, even though the functionality is not useful or usable without the necessary non-functional characteristics”*, L. Chung et al., “On Non-Functional Requirements in Software Engineering”, 2009 [62].

Software Quality

According to M. Glinz [98], functional and non-functional requirements set the boundary of an important dimension of the software, its quality. For example, in the ISO/IEC 25000, also known as SQuaRE[1] [116] quality standard we can find Quality Attributes (QAs) such as: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. Notice that these quality characteristics are the same that were mentioned in the previous definition of the term NFR. In other words, NFRs and software quality are highly related.

> *“[Software quality is...] the capability of software product to satisfy stated and implied needs when used under specified conditions”*, ISO/IEC 25000, 2005 [116].

The *needs* mentioned in the previous definition could be any kind of software requirement: functional, or non-functional. It is important to notice that using the term quality as defined in the ISO/IEC standard refers to the grade of satisfaction of both types of requirements: functional and non-functional. But in some communities is common to refer to NFRs as Quality Requirements (QRs).

[1]SQuaRE was published in 2005 as a substitution of the ISO/IEC 9126 [117]

1.1.2 Software Architecture

Software architecture is the result of the Architectural Decisions[2] (ADs) made during the architecture design. ADs and their recognition as first-class elements are one of the most important advances in software architecture during the last decade. One definition of software architecture is:

> *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."*, L. Bass et al., "Software Architecture in Practice (second edition)", 2003 [36].

NFRs have been recognized as one of the main drivers to make ADs. For example, the selection between several pieces of technology, the decision to replicate some components, or even the mere existence of some component as part of the architecture. What criteria is used to choose among several technologies? Why some component needs to be replicated? The typical answer to these questions is a NFR. These quotes are in this direction:

> *"[NFRs...] play a critical role during system development, serving as selection criteria for choosing among myriads of alternative designs and ultimate implementations"*, L. Chung et al., "Non-Functional Requirements in Software Engineering", 2000 [63].

> *"The rationale behind each architecture decision is mostly about achieving certain NFRs"*, L. Zhu and I. Gorton, "UML Profiles for Design Decisions and Non-Functional Requirements", 2007 [217].

Architectural Knowledge

Architectural decisions are the base of Architectural Knowledge (AK). There are many definitions of AK (see [77]), the simplest being: *"Architectural Knowledge = Design Decisions + Design"* [145].

[2]Also referred in this thesis as *architectural design decision* and *design decision*.

> *"Architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is."*, P. Kruchten et al., "Building Up and Reasoning About Architectural Knowledge", 2006 [145].

This knowledge is not easily accessible, normally it resides in the architect's mind, and in few cases it is disseminated in the project documentation, but even in this case it lacks of explicit reasoning about the alternatives considered previous to the decision.

> *"Except for the architecture design part, most of the architectural knowledge usually remains hidden, tacit in the heads of the architects"*, P. Kruchten et al., "Building Up and Reasoning About Architectural Knowledge", 2006 [145].

1.1.3 Model-Driven Development

Software engineering researchers have been trying to find ways to systematize the software development processes. A software development paradigm that is gaining relevance in the last years is MDD, a development paradigm where models (and their transformation) play a fundamental role [28, 155]. MDD is based on the separation of the essential specification of the system and its implementation using a specific platform. In MDD, models are used to specify, simulate, verify, test and generate the system to be built. The benefits of using MDD are higher abstraction level and improved platform independence. A clear example of these benefits could be the adaptation to new technologies; this problem can be alleviated by using technology-independent models that can be transformed semi-automatically into technology-specific models to fulfill the trendy technological needs. MDD is defined as:

> *"Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing"*, S. Mellor et al., "Model-Driven Development", 2003 [155].

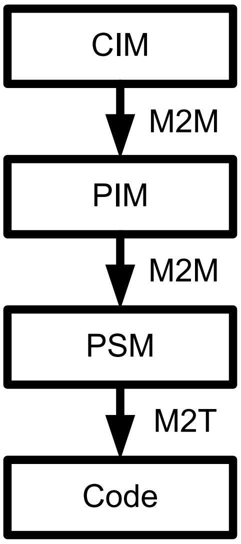


Figure 1.2: Models and transformations of MDD

In other words, MDD uses models as the primary artifact of the software production process, and development steps consist of the application of transformations over these models. Due to its promised benefits, MDD is being one of the main issues of organizations such as OMG, and is also mentioned as a driver in particular types of systems (e.g., for self-adaptive systems [59]).

Normally, MDD approaches focus on conceptual models, the implementing technologies, and the generation of code. One of the most popular MDD approach is the Model-Driven Architecture (MDA), an OMG standard [157], that has been used as the basis for many other later MDD approaches.

MDA distinguishes several types of models and transformations:

Computation Independent Models (CIM) also called *domain model*, does not show details of the structure of systems.

Platform Independent Models (PIM) represents the software system without considering the technology platform used.

Platform Specific Models (PSM) is a refined version of the PIM with details of the technological platform, i.e., two different implementations of the same system would share the same PIM but have two different PSMs, each one adapted to the technological capabilities of each platform.

Model-to-Model (M2M) transformations evolve one of these types of models to a different type (vertical transformation) or the same type (horizontal transformation).

Model-to-Text (M2T) transformations are normally used to generate the code or the documentation of system from the PSM (but in practice may apply to other ends). These transformations include generating several code artifacts glued together, e.g., Java business classes, Oracle DB schemes, etc.

These models and transformations have become the *de facto* standard in MDD approaches (see Figure 1.2).

1.2 Research questions

> *"[Engineering is...] the creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property"*, Encyclopedia Britannica[3].

Software architecture, as an artifact, reflects many of the aspects highlighted in the engineering definition. A software architecture captures the design, normally as a combination of components, and forecasts its behavior for particular aspects (specially the non-functional). Considering this, software architecture should be a fundamental pillar and a central artifact of any software engineering process.

In the particular case of MDD (see Section 1.1.3), architecture is not a fundamental part of the engineering process. Still, architecture is always

[3]`http://www.britannica.com/EBchecked/topic/187549/engineering`

Table 1.1: Research questions of this thesis

Id	Research Question
RQ1	How NFRs and architecture can be integrated in the MDD process?
RQ2	How do NFRs impact on architectural design?
RQ3	Which AK is necessary to make architectural decisions?

there, but as something predefined and hard to adapt. Most of the enthusiasm around MDD was about providing an automatic software development, in this situation aspects that require creativity and decision-making such as NFRs and architecture have been hidden from the process by giving them default values. In contrast, in Model-Driven Engineering (MDE) the idea of automatic software development is not so important. MDE opens the door to handle NFRs in MDD and the introduction of architecture design as part of the MDD process. Having identified this necessity in MDD to evolve to an engineering process, the first research question (RQ1) of this thesis is: *How NFRs and architecture can be integrated in the MDD process?*

The answer provided to RQ1 is an extension to the MDD process [24] which is explained in Chapter 3. This extension is presented as a framework that integrates NFRs in the MDD process and we also explore different issues such as the need of NFRs formalization, and the need of inclusion of architectural models in the MDD methodology. As consequence of the integration of NFRs into MDD, architectural design is proposed to be part of this development process.

As result of RQ1, the architectural design proposed as part of the MDD process aims to a computer assisted method to help architects in the architectural decision-making using NFRs as drivers of this method. As consequence, the second research question (RQ2) is: *How do NFRs impact on architectural design?*

At the same time, the decision-making method that will is surfaced from the result of the RQ1 and RQ2 requires Architectural Knowledge (AK), which have to be defined, represented, and tested. As consequence, the third research question (RQ3) is: *Which AK is necessary to make architectural decisions?*

The list of research questions is shown in Table 1.1.

Table 1.2: Shaw's list of research settings

Research setting	Sample question
Feasibility	Is there an X, and what is it? Is it possible to accomplish X at all?
Characterization	What are the important characteristics of X? What is X like? What, exactly, do we mean by X? What are the varieties of X, and how are they related?
Method/Means	How can we accomplish X? What is a better way to accomplish X? How can I automate doing X?
Generalization	Is X always true of Y? Given X, what will Y be?
Selection	How do I decide between X and Y?

1.3 Methodological approach

Shaw provides several ways of characterizing software engineering research, in terms of what she describes as research settings, research products, and validation techniques [195].

Research settings are the different classes of research problems. Shaw lists five research settings along with a sample question as example (see Table 1.2). The settings of this thesis, in terms of Shaw's characterizations, are characterization, and method/means. RQ1 tries to find *means* to include NFRs and architecture in MDD, RQ2 is about the *characterization* of the relationship between NFRs and ADs, and RQ3 is clearly the *characterization* of architectural decisions.

Research products are the *tangible* results of the research project. Shaw lists five research products along with a short description of how to achieve it (see Table 1.3). The research products of this thesis include qualitative or descriptive model, technique, and analytic model. *Descriptive models* are the empirical studies carried to understand the practice of software architects with regard to NFRs and architectural decisions, and the chapters dedicated to the state of the art. The *technique* produced is the MDD process extended to include NFRs and architectural models. The *analytic model* is an ontology designed to manage the architectural knowledge.

The last characterization is the research validation. Shaw provides a list of five validation techniques (see Table 1.4). The validation techniques used in this thesis are persuasion, implementation, evaluation, and experience.

Table 1.3: Shaw's list of research products

Research product	Research approach or method
Qualitative or descriptive model	Organize and report interesting observations about the world. Create and defend generalizations from real examples. Structure a problem area; formulate the right questions. Do a careful analysis of a system or its development.
Technique	Invent new ways to do some tasks, including procedures and implementation techniques. Develop a technique to choose among alternatives.
System	Embody result in a system, using the system development as both source of insight and carrier of results.
Empirical predictive model	Develop predictive models from observed data.
Analytic model	Develop structural (quantitative or symbolic) models that permit formal analysis.

Table 1.4: Shaw's list of validation techniques

Technique	Character of validation
Persuasion	A technique, design or example.
Implementation	Of a system or technique.
Evaluation	With respect to a descriptive model, a qualitative model, an empirical quantitative model.
Analysis	Of an analytic formal model, an empirical predictive model.
Experience	Expressed in a qualitative or descriptive model, as decision criteria or an empirical predictive model.

Persuasion is used all along the thesis using examples that illustrate the behavior of the proposed ideas or processes. For the *implementation* technique, a tool has been developed to show the feasibility to use the proposed ontology and the method to assist architects in architectural decision-making. The *evaluation* of the data gathered from the empirical studies, was contrasted with the results obtained in similar studies (ours and others'). The design of the ontology, the method, and the tool were based on the *experience* obtained from the interviewed architects.

1.4 Research contributions

Having established the research questions, the published works related to this thesis can be grouped in three areas: integration of NFRs and architecture into MDD (RQ1), empirical research around software architecture and NFRs (RQ2), and research around the AK (RQ3).

The most important contributions of this thesis have been published in recognized venues such as the *IEEE Software*, and *Journal of Software: Practice and Experience (SPE)*, *IEEE International Requirements Engineering Conference (RE)*, *European Conference on Software Architecture*, and *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Also, some works have been published in specialized workshops such as *TOPI*, a workshop, held in the *International Conference on Software Engineering*, for research around tools as plugins (which is the case of the implemented tool to manage architectural knowledge); *IWSSA*, a workshop specialized in software architecture; *EASA*, which is specialized in empirical works done around software architecture; and *DSDM*, a Spanish MDD workshop. I also mention the bachelor thesis and the master thesis because they represent important milestones for the research done in this thesis.

1.4.1 Integration of NFRs into MDD

The origin of this PhD thesis is the bachelor thesis [8], in this work we had a UML profile for class diagrams used to identify responsibilities (e.g., the identifier attribute of a particular class) and then choose a treatment (e.g., use a primary key in a database schema) to handle this responsibility. This work evolved to a more complex framework, that was published in the research in progress track of the *33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* [17] and in the *Desarrollo de Software Dirigido por Modelos (DSDM)* [16] workshop. As the framework evolved the complexity became a problem because responsibilities and treatments were too fine-grained. To solve this situation, responsibilities and treatments became coarse-grained, turning into NFRs and architectural decisions respectively. This change coincided with the presentation of the master thesis [9] where a new version of the framework was released. After some maturation of the ideas behind this framework, the work was pub-

Table 1.5: Works related to the integration of NFRs into MDD

Ref.	Pub. Type	Venue	Year	Title
[8]	Bachelor thesis		2007	Assignació de responsabilitats a capes usant AndroMDA[a]
[17]	Conference (short paper)	Euromicro	2007	Assigning Treatments to Responsibilities in Software Architectures
[16]	Workshop	DSDM	2007	Asignación de Tratamientos a Responsabilidades en el contexto del Diseño Arquitectónico Dirigido por Modelos[b]
[9]	Master thesis		2009	Considering Non-Functional Requirements in Model-Driven Engineering
[24]	Conference (full paper)	RE	2010	Dealing with Non-Functional Requirements in Model-Driven Development

[a] English translation: Setting responsibilities into layers using AndroMDA
[b] English translation: Setting treatments to responsibilities in the MDD context

lished with the collaboration of Jordi Cabot in the *18th International IEEE Requirements Engineering Conference (RE)* [24] as a vision paper on how to integrate NFRs into MDD. Table 1.5 lists the published works related to the integration of NFRs into MDD in chronological order.

1.4.2 Empirical research in software architecture

In this thesis we have executed three empirical studies:

- As part of the master thesis [9], there was a protocol to drive an electronic survey about the industrial practice of software development. The results of this study were presented, in the *1st Empirical Assessment in Software Architecture (EASA)* [20] workshop, in this case focusing on the results related to software architecture, and in the *16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)* [21] focusing on the results related to NFRs.

- After this first experience, it was necessary to go deeper into the details. In collaboration with Claudia Ayala and Jordi Cabot, we started a new empirical study based on individual interviews about how architects deal with NFRs in their projects, this study was published in the *20th IEEE International Requirements Engineering Conference* [11]. In response to this study we collaborated with Frank Buschmann in

Table 1.6: Works related to empirical research in software architecture

Ref.	Pub. Type	Venue	Year	Title
[9]	Master thesis		2009	Considering Non-Functional Requirements in Model-Driven Engineering
[20]	Workshop	EASA	2009	Usage of architectural styles and technologies in IT companies and organizations
[21]	Conference (ext. abstract)	REFSQ	2010	How do Software Architects consider Non-Functional Requirements: A Survey
[11]	Conference (full paper)	RE	2012	How do Software Architects Consider Non-Functional Requirements: An Exploratory Study
[50]	Journal (column)	IEEE Software	2012	Architecture Quality Revisited
[12]	Journal	IEEE Software	2013	Non-Functional Requirements in Architectural Decision-Making
[25]	Conference (short paper)	ECSA	2013	The Role of Quality Attributes in Service-based Systems Architecting: A Survey

writing a column in the *IEEE Software* magazine [50] highlighting some of the most relevant outcomes of the study. After these two last publications we got accepted a research paper to the *IEEE Software* magazine but this time focusing on the architectural decisions [12].

- The last empirical study was produced as result of a research stay in the Groningen university and in collaboration with Matthias Galster and Paris Avgeriou. In this case the study was focused on a particular architectural style: Service-Oriented Architecture, and we studied how quality attributes affect architectural decisions. This study was published in the *European Conference on Software Architecture* [25].

Table 1.6 lists the published works related to empirical research in software architecture in chronological order.

1.4.3 Architectural knowledge

For this part of the research there are three parallel lines of action:

- The design of an ontology to manage architectural knowledge. The ontology, called Arteon, was published in *DSDM* [19] as an initial version, and afterwards the two separated modules that compose this ontology were presented in more detail, one centered in the structural part of the architecture, published in the *9th International Workshop on System/Software Architectures (IWSSA)* [22] and the other centered in the architectural decisions and the reasoning process, published in the *XVI Congreso Iberoamericano en Ingeniería de Software (CIbSE)* [23]. In this thesis is presented the current state of the ontology, which is the aggregation of the last published versions.

- The design of a method to assist architects in the architectural decision making. The method, called Quark, was presented in the *XVI Congreso Iberoamericano en Ingeniería de Software (CIbSE)* [23]. This method is based on the architects' feedback obtained from the empirical studies, and in the Arteon ontology.

- The design of a tool to manage and reuse the architectural knowledge. The tool, called ArchiTech, was published in the *1st Workshop on Developing Tools as Plug-ins (TOPI)* [13], this paper was invited for an extended version in the *Software: Practice and Experience* journal [15]. Both papers (TOPI and SPE) explained the part of the tool dedicated to the management of architectural knowledge (this part is based on the Arteon ontology), the part dedicated to reasoning and making architectural decisions (based in Quark) was presented, as tool demonstration, in the *20th IEEE International Requirements Engineering Conference (RE)* [14].

Table 1.7 lists the published works related to the ontology for architectural knowledge in chronological order.

1.4.4 Other publications

There have been other published works that are not related to the main topic of this thesis.

- In 2008, we designed the architecture of a SOA monitoring system called SALMon. This architecture was published in the *7th IEEE International Conference on Composition-Based Software Systems (ICCBSS)* [18]. The experience gained in Service-Oriented Architecture (SOA) during the design of SALMon has been used many times as

Table 1.7: Works related to the ontology for architectural knowledge

Ref.	Pub. Type	Venue	Year	Title
[19]	Workshop	DSDM	2009	Definición de una Ontología para el Proceso de DSDM considerando Requisitos No-Funcionales[a]
[13]	Workshop	TOPI	2011	Reconciling the 3-layer Architectural Style with the Eclipse Plug-in-based Architecture
[22]	Workshop	IWSSA	2011	Ontology-based Architectural Knowledge representation: structural elements module
[15]	Journal	SPE	2012	The Three-Layer Architectural Pattern Applied to Plug-in-based Architectures: the Eclipse Case
[14]	Conference (tool demo)	RE	2012	ArchiTech: Tool Support for NFR-Guided Architectural Decision-Making
[23]	Conference (full paper)	CIbSE	2013	Quark: a method to assist software architects in architectural decision-making[b]

[a] English translation: Defining an ontology for the MDD process considering NFRs
[b] This paper was finalist for the best paper award

examples of architectural decisions or to identify the key concepts of the Arteon ontology. Later in the same year, Marc Oriol took the lead of the SALMon project, and we made a joint publication with the advances of SALMon during 2008 [172].

- In 2011, the research group (GESSI[4]) started a new research project about requirements engineering in the context of service oriented systems. In the first year of this project we collaborated in a vision paper of the work plan of this project [189].

- in 2012, GESSI started a collaboration between the university (UPC) and a software consultancy company (everis). In this collaboration we performed an empirical study about reference architectures. The design on this study was presented in 2013 in the *10th Experimental Software Engineering Track Workshop (ESELAW)* [150]. In this study we contributed with the experience obtained driving empirical studies to software architects (e.g., helping in the design of the questionnaires and performing the interviews).

[4] `www.essi.upc.edu/~gessi`

Table 1.8: Works that are not directly related to this thesis

Ref.	Pub. Type	Venue	Year	Title
[18]	Conference (short paper)	ICCBSS	2008	Service Level Agreement Monitor (SALMon)
[172]	Workshop	MONA+	2008	Monitoring Adaptable SOA-Systems using SALMon
[189]	Conference (full paper)	JCIS	2011	Ingeniería de requisitos orientada a servicios: características, retos y un marco metodológico[a]
[150]	Workshop	ESELAW	2013	A Framework for Software Reference Architecture Analysis and Review[b]
[92]	Conference (position paper)	ICSOFT	2013	Managing Risk in Open Source Software Adoption

[a] English translation: Service Oriented Requirements Engineering: Characteristics, challenges, and a methodological framework
[b] This paper got the best paper award

- In 2013, GESSI started an European project, RISCOSS[5]. This project had several tasks related to requirements elicitation and architectural design in which we could use much of the experience obtained during this thesis. As preliminary result of this project we have published a paper with the vision and objectives of RISCOSS [92].

Table 1.8 lists the published works not directly related to this thesis in chronological order.

1.4.5 Statistics of the published works

Excluding the publications not related to this thesis, in Figure 1.3-left we can see in chronological order, and separated by the three research topics, the publications related to this thesis. We can observe that MDD research finished in 2010, and that the empirical works and the ones related to architectural knowledge were carried in parallel after the MDD research. Also, we can see that during the time of this thesis there was an average of more than 2 publications per year.

Counting the publications by types, as shown in Figure 1.3-right, we can see that the published works are 3 times in journals, 7 times in conferences, and 5 times in workshops. In most venues, there have been one publication,

[5]`www.riscoss.eu`

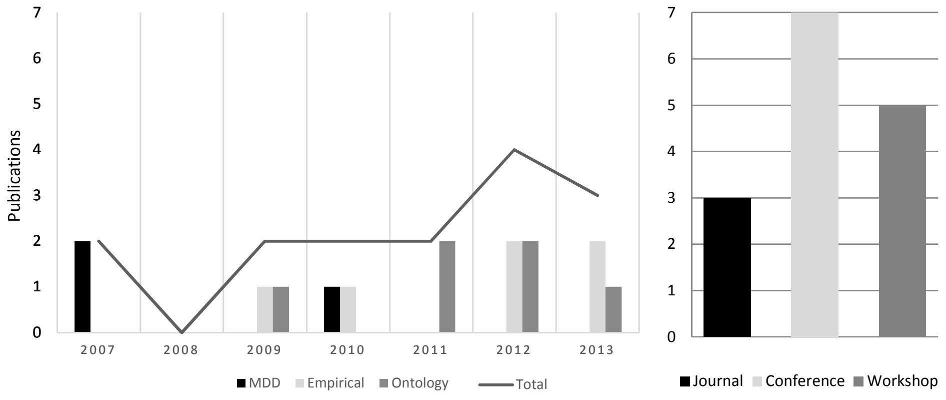


Figure 1.3: Statistics of published works

with the exception of RE conference (three times), IEEE Software journal (twice), and DSDM workshop (twice).

Table 1.9 shows the number of citations of the published works in journals, conferences, and workshops related to this thesis. The number of citations were obtained from the Google Scholar records in the 18th of November, 2013. We also added as remarks, the CORE ranking[6] for conferences, and the impact factor for journals[7]. It is worth to remark that some publications are still too recent to have a clear idea of the amount of citations that they will get.

1.5 Structure of the dissertation

This thesis is presented in three parts, which corresponds to the three research questions exposed in Section 1.2. The first part, *"Non-Functional Requirements in Model-Driven Development"*, refers to RQ1, the second part, *"Non-Functional Requirements in Software Architecture"*, refers to RQ2, and the third part, *"Arteon, Quark, and ArchiTech"* refers to RQ3. Each part begins with a chapter of the state of the art of the related research topics, followed by the contributions of this thesis (see Table 1.10).

[6]We used the last available conference ranking, February 2008 (`core.edu.au`).
[7]We used the ISI Journal Citation Reports, 2011.

Table 1.9: Number of citations of published works related to this thesis

Citations	Ref.	Topic	Venue	Year	Remarks
31	[24]	NFRs into MDD	RE	2010	CORE A
10	[11]	Empirical research	RE	2012	CORE A
8	[21]	Empirical research	REFSQ	2010	CORE B
4	[22]	Architectural Knowledge	IWSSA	2011	Workshop
3	[12]	Empirical research	IEEE Soft.	2013	I.F.: 1.508
3	[16]	NFRs into MDD	DSDM	2007	Workshop
2	[14]	Architectural Knowledge	RE	2012	CORE A
2	[17]	NFRs into MDD	Euromicro	2007	CORE C
2	[13]	Architectural Knowledge	TOPI	2011	Workshop
1	[15]	Architectural Knowledge	SPE	2012	I.F.: 0.519
1	[19]	Architectural Knowledge	DSDM	2009	Workshop
1	[20]	Empirical research	EASA	2009	Workshop
0	[50]	Empirical research	IEEE Soft.	2012	I.F.: 1.508
0	[23]	Architectural Knowledge	CIbSE	2013	Not ranked
0	[25]	Empirical research	ECSA	2013	CORE A

Table 1.10: Summary of chapters of this dissertation

Part	Chapter	Type	RQ	Source of contributions
	1	Introduction		
I	2	State of the art		[9]
I	3	Contribution	RQ1	[24]
II	4	State of the art		[11, 25]
II	5	Contribution	RQ2	[20, 21, 11, 12, 25]
III	6	State of the art		[10]
III	7	Contribution	RQ3	[22, 23, 15, 14]
	8	Conclusions		

Part I

NFRs in Model-Driven Development

Chapter 2

State of the art

This state of the art is based on the state of the art presented in the master thesis [9], and has been updated with relevant works published after the master thesis.

One side of the research done in MDD is driven by the industry, this part of the research is oriented to frameworks (e.g., Eclipse[1]), and standardization (e.g., Unified Modeling Language, UML). On the other side, the academic research is oriented to new and experimental development approaches, and solutions to very particular problems. This state of the art is centered in the academic side of contributions that bring NFRs support to MDD. Two topics are included:

- *Modeling languages that support NFRs in MDD.* There are two strategies to model NFRs in MDD, one is to extend an existing language with NFRs support (e.g., use a UML profile), and the other one is to give MDD support to a language thought to represent NFRs (e.g., the $i*$ framework [214]).

- *Processes that support NFRs in MDD.* One of the most well-known approach in industry is MDA [157], but there are many other MDD processes, normally to solve or alleviate a concrete problem.

[1]`www.eclipse.org`

This state of the art is based on searches in research databases (e.g., Web of Knowledge[2]). Some techniques mentioned in the guidelines proposed by Kitchenham [131] were used to improve the quality of this study. Also some papers were included as suggested by experts of the area (e.g., conversations with other researchers while attending conferences and workshops and research stays in other universities).

2.1 Modeling languages supporting NFRs in MDD

As said before, for the representation of NFRs in models, in the MDD context, there are two perspectives. The extension of the current modeling languages used in MDD to represent NFRs or adapt the MDD process by adding support to the requirements engineering specific modeling languages.

2.1.1 Extending modeling languages to represent NFRs

In this perspective NFRs are supported by extending the current modeling languages used in MDD. The most used modeling language in MDD is UML, which has UML profiles as the standard extension mechanism (in a recent study, not published yet, we found that the 43.3% of papers with a NFR-aware MDD approach to develop Service-Oriented Architectures use UML Profiles). Other MDD approaches use Domain Specific Languages (DSLs), these modeling languages are lightweight and focused into one particular domain by providing specific abstractions and notations (e.g., a networking DSL may have a WiFi abstraction with the typical WiFi logo as notation). In this study we focus on DSLs that have been extended to support NFRs.

The following are some important NFRs-aware UML profiles:

- The profile named Modeling and Analysis of Real-time and Embedded systems (MARTE) [169]. This profile, adopted by the OMG organization, supports the representation of NFRs with measurement sources, precision, and time expressions.

- UML profile for Quality of Service and Fault Tolerance (QoS-FT) [170]. This profile, adopted by the OMG organization, uses two types of an-

[2]`www.webofknowledge.com`

notations: instantiation of template classes from the QoS Catalogs and annotations in UML models with QoS Constraints and QoS Values.

- S. Bernardi et al. [40] present an extension to MARTE to give support for dependability analysis. Another profile that explores dependability is DMP [64].
- H. Wada et al. [211] uses a UML profile, named UP-SNFR, to support the specification of NFRs inside UML models for SOA systems. It is based on the idea of features.
- L. Zhu and I. Gorton [217] used UML profiles to specify architectural decision and NFRs, the good thing is that they can be used together. L. Zhu and Y. Lin [218] continued this work about modeling non-functional aspects and their impact on design decisions.
- J. Jürjens [124, 125] proposed a UML profile for Security (UMLsec).

The following are DSLs extended with NFR support:

- L. Gönczy et al. [99] presented an approach for performance analysis in the context of SOA systems. The approach uses its own metamodel to represent NFRs.
- S. Kugele el al. [146] uses its own metamodel in the context of safety-critical real-time systems for embedded systems. In this case the treated NFR is resource usage (e.g., memory, processing time, etc.).
- F. Molina and A. Toval [158] presented an approach to integrate usability in MDD in the context of Web information systems. In this case the metamodel contemplates functional and non-functional requirements.

2.1.2 MDD with support for RE modeling languages

In the second perspective, the MDD process is adapted to support requirement engineering models. NFRs are a matter of study of RE, and in consequence they are normally considered in the modeling languages of this field. Researchers of RE have proposed different notations that handle NFRs, the most widespread being: the NFR framework [63], the *i** framework [214], KAOS [210], and Problem Frames [119]. Examples for these notations may be found in [200].

All the papers found that integrate RE modeling languages in MDD belong to Goal-Oriented Requirements Engineering (GORE). GORE is an approach that advocates for the identification and analysis of goals as a prerequisite for writing complete and consistent requirements documents. J. Cabot and E. Yu [52] have exposed their ideas about the directions to take, and the open problems to support NFRs with goal modeling in MDD. These are some works that use GORE in the context of MDD:

- J. Mazón et al. [152] use *i** models to model goals and requirements, and then derive a conceptual multidimensional model to support the decision making process of designing a data warehouse. There is no special mention of NFRs, but soft goals are supported in the *i** metamodel used.
- S. Konrad et al. [136] use goal models as defined in thel NFR Framework [63]. NFRs are modeled with information that indicates if the NFR helps or hurts a particular alternative pattern.
- A. Fatwanto and C. Boughton [88] start from a NFR Framework [63] model and then use UML diagrams with annotations. The paper exemplifies the use of the approach with a security related scenario.
- F. Alencar et al. [3] integrates GORE and MDD to fill the gap between requirements specification and the implementation of the software product. The approach is focused on functional requirements, but uses *i** models for requirements. The work presented is framed into the OO-Method [176].
- J. Castro et al. [57, 148] use *i** models to model requirements, then produce an architectural model. In these works there are specific mentions to the support of NFRs and is based on the product lines principles. The produced architectural model are described using ACME [94]. More recently this approach has been adapted to produce documented architectural decisions [81].

Curiously, no relevant paper using Service-Level Agreements (SLAs) to drive the development of Service-Oriented Architectures (SOA) was found

Table 2.1: Comparison of modeling languages that support NFRs in MDD

Ref.	Types of NFRs	Domain	Notation type
[169]	Measurable constraints	Real-time systems	UML profile
[170]	Quality of Service	Any	UML profile
[40, 64]	Dependability	Real-time systems	UML profile
[211]	Security, Fault Tolerance	Service Oriented Architecture	UML profile
[217, 218]	Any	Any	UML profile
[124, 125]	Security	Any	UML profile
[99]	Performance	Service Oriented Architecture	DSL
[146]	Resource usage	Embedded systems	DSL
[158]	Usability	Web Information Systems	DSL
[152]	Any	Data Warehouses	GORE
[136]	Any	Any	GORE
[88]	Operationalizable NFRs	Any	GORE
[3]	Functional Req.	Information Systems	GORE
[57, 148]	Any	Any	GORE

in the literature. This is curious because, SLAs are the natural way to express the desired Quality of Service (QoS), which may be understood as non-functional requirements.

2.1.3 Analysis

The Table 2.1 summarizes the works about notation of NFRs in models. We can see in this table that UML profiles and DSL are mostly designed for specific purposes (concrete type of requirements or a particular domain) while GORE approaches try to embrace requirements in a more generic way.

In the present thesis NFRs are related to design decisions, as explained in Section 7.1, this link is also suggested by L. Zhu and Y. Lin [218].

2.2 Processes that support NFRs in MDD

In the introduction we have argued that NFRs have an important effect in the final form that the software system takes. If we consider MDD, we may say that an optimal MDD process should be able use the elicited NFRs to select and apply the most adequate transformations, in order to generate a software system that satisfies the desired NFRs. In this section we investigate to what extent this need is currently fulfilled.

NFRs may play two different non-exclusive roles inside MDD. The first is to use NFRs for driving the transformations and the second is to use NFRs to validate the results.

2.2.1 NFR-driven transformations

In these works NFRs act as the selector of the behavior of the transformation. Some of these works are called as *Quality-driven transformations*, in the introduction we have seen that quality is composed by functional and non-functional factors. This is especially true when we look at quality-driven transformations which in some cases are referring more to the functional part of the quality. The works found related to this topic are:

- A. Solberg et al. [197] presented an approach that use the most adequate transformations depending on the QoS requirements. The approach presented in the paper is based on gradually resolving QoS requirements when a model transformation is performed. These transformations use patterns that improve some aspect of QoS.
- A. Sterritt and V. Cahill [199] presented an approach to make M2M transformations that take NFRs as parameters. The target model in this case is an architectural model. Concretely, in this paper they deal with distribution issues of architectural styles.
- J. I. Panach et al. [175] presented an approach to tackle with usability. Only the functional usability features are part process to drive the transformations. The work presented is framed into the OO-Method [176].

2.2.2 NFRs as validating mechanism

The approaches that use NFRs to act as validating mechanism use the MDD techniques to generate models with specific formalism where NFRs can be analyzed. These are the most representative works in this direction:

- S. Röttger [188] proposed, in the context of Component-Based Software Engineering, a method/tool that generates measurable models for Quality of Service (QoS). For example, the model is extended with notations such as "response time value < 500" in a class stereotyped as "NFR".

- G. Rodrigues et al. [186] proposed a method to validate the reliability by using prediction models. They defined a profile for reliability analysis based on scenario specifications by extending UML. This notation is transformed into labeled transition system (LTS) for the prediction analysis.

- V. Cortellessa [72] presented a framework that uses NFRs to validate the resulting models. This approach proposes a separation of NFRs into the same levels of abstraction as MDA to perform the validation in each level of the development process. As depicted in the paper, there will be a model for each type of NFR, a part from the typical model to represent the functionality.

- A. Fernandez et al. [89] proposed an approach to include a usability model as part of the MDD process for Web development. This model is evaluated using some metrics to produce system usability reports, the usability model is based in the ISO/IEC 25000 [116]. A similar method/tool has been done by F. Molina and A. Toval [158], in this case, they propose a metamodel to represent NFRs related to usability in the web information systems domain and use the quality model and metrics proposed in [53].

- D. Ardagna et al. [27] extended V. Cortellessa's work [71] considering run-time validation of performance and reliability types of NFRs. D. Ardagna et al. proposes to use models not only in development, but also in run-time. This idea brings the possibility measure the NFRs and adapt the software system as required. This is especially relevant in service oriented approaches.

- S. Gallotti et al. [93] proposed an approach/tool that checks the QoS in SOA using probabilistic model checking. The model analyzed is generated during the MDD process. In this approach they use MARTE [169] as NFR notation.

Table 2.2: Comparison of approaches that consider NFRs in MDD

Ref.	Types of NFRs	Domain	Instrument	Category
[197]	Quality of Service	Any	Patterns	NFR-driven transformations
[199]	Any	Any	Patterns	NFR-driven transformations
[175]	Usability	Any	OO-Method	NFR-driven transformations
[188]	Quality of Service	Any	Measurable models	Analysis and validation
[186]	Reliability	Any	LTSA[a]	Analysis and validation
[72]	Any	Any	Independent	Analysis and validation
[71]	Performance and reliability	Any	PRIMA[b]and COBRA[c]	Analysis and validation
[158, 89]	Usability	Web IS	Quality metrics	Analysis and validation
[27]	Performance and reliability	Any	Markov chains	Analysis and validation
[93]	Performance and reliability	SOA	Probabilistic models	Analysis and validation

[a] Labeled Transition Systems Analyzer.
[b] PeRformance IncreMental vAlidation.
[c] COmponent-Based Reliability Assessment.

2.2.3 Analysis

The Table 2.2 summarizes the MDD approaches that consider NFRs. We can see in this table that the group of works based in NFR-aware transformations, and in particular the approach proposed by A. Sterritt and V. Cahill [199], are in the same direction as this thesis. More details are explained in Chapter 3. The quality-driven works found are more oriented to fulfill functional aspects of the quality than non-functional ones, and the group of analysis and validation is where there are more works. This last group of works complement the work presented in this thesis, it could be possible to integrate both approaches together (NFR-aware transformations and analysis and validation) with an improvement in the results.

Chapter 3

Introducing NFRs in MDD

This chapter is based on the main contributions of [24].

The impact of NFRs over software systems design has been widely documented (see Chapter 1). Consequently, cost-effective software production methods shall provide means to integrate this type of requirements into the development process. The state of practice (explained in Section 3.1) shows that current MDD approaches do not tackle NFRs satisfactorily and that limits their success and applicability, and hampers their adoption by the industry.

In this chapter we analyze this assumption over a particular type of software production paradigm, MDD, and we outline a general framework that smoothly integrates NFRs into the core of the MDD process and provide a detailed comparison among all the MDD approaches considered. In this integration, software architecture emerges with a predominant position. To motivate our findings we use an academic exemplar about the development of a web portal for a travel agency. Finally, we identify some research issues related to this framework.

3.1 State of the practice

There are a great variety of MDD-based approaches, many of them following the two-level (PIM and PSM) transformation introduced in the OMG's MDA approach [157]. Among the most popular ones, we find the Executable UML proposals, with [154] as the most popular representative. Executable UML uses a reduced subset of UML that it is directly executable, either using UML interpreters or by providing a direct translation from the models to the final code. Some action is required in this critical situation in order to make the MDD approach more effective, even more considering that this Executable UML method [154] is the basis for the new OMG standard "Semantics of a Foundational Subset for Executable UML Models" [171] that pretends to increase the use of UML in a MDD context.

The adoption of MDD techniques is being slow [204], even that some industrial studies suggest that they increase the productivity [139, 198]. NFRs were not mentioned in the comparison of case studies provided by Krogmann et al. [139], but Staron [198] mentioned several times that the quality assurance, and validation of quality in models, are conditions for the adoption of MDD in industry.

To understand how developers deal with NFRs when using MDD approaches we consulted experts in MDD to explain their experiences and observations. NFRs exist independently of being explicit in the process, and at some point the developer should deal with them. If we consider the general form of MDD (see Figure 1.2), we may envisage two different, non-exclusive approaches to make a generated product compliant with NFRs even if they are not represented as part of the MDD process (as we have seen in the academic approaches presented in Section 2.2): NFRs supported with manual adaptation and NFRs supported with new transformations. But if we look at the real limitations of the tools we can get even a worse scenario, which is described later in this section.

3.1.1 NFRs supported with manual adaptation

The software developer directly modifies by hand the result of the MDD process (see Figure 3.1-a). In its simplest form, the developer directly modifies the code obtained after the final M2T transformation. In the best case, the developer will able to work at the PSM level, modifying the model to adapt it to the NFRs, and then use the M2T transformation (possibly modified somehow) to generate the code. For example, in an empirical study that interviewed several companies that use MDE [114] there was a very clear quote from one of its respondents of this kind of adaptation: *"So after gen-*

erating the code, if you want to do this then you have to delete this line and that line and also you have to change this parameter like this."

This manual adaptation of the system collides with the essence of the MDD paradigm and has several drawbacks:

- Takes longer to produce the software.
- Provokes lower reliability of the final product due to the human-based post-process.
- Damages traceability and thus comprehension.
- In case of changes due to maintenance, either the post-process has to be replicated or the maintenance is directly made on the final product.

3.1.2 NFRs supported with new transformations

The MDD engineer modifies the M2M transformation in order to obtain a PSM that satisfies the NFRs (see Figure 3.1-b). For example, we could have several transformations for producing PSM compliant with different strategies that satisfies different sets of NFRs. The drawbacks above are therefore solved, but others appear in their place:

- The complexity of the MDD framework is greater, because there are more transformations to maintain.
- It is difficult to anticipate all the possible scenarios, in fact it may be even impossible (e.g., replication strategies may be applied in many different ways, and each would require a different transformation).
- The selection of the most appropriate transformation (for the given set of NFRs) to apply relies on the software architect, becoming a human-based pre-process, incrementing thus the likelihood of errors in decision-making.
- When the software architect realizes that the available transformations are not adequate for the current process it is necessary to build a new ad-hoc one, making the initial configuration time longer.

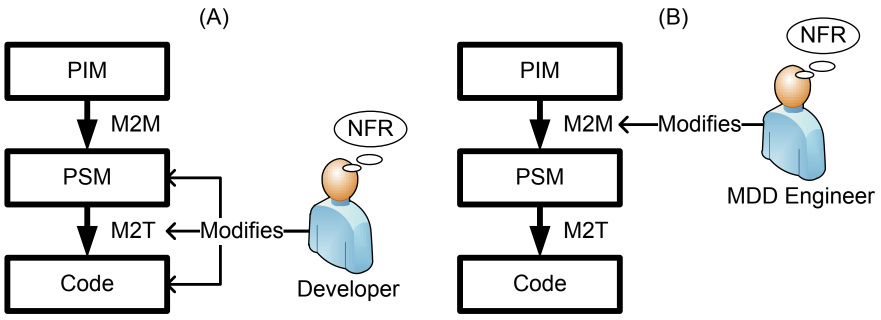


Figure 3.1: Dealing with NFRs in a classical MDD approach

3.1.3 How are NFRs supported in MDD practice?

The two approaches presented above represent two extreme cases. Hybrid solutions may also exist, where some NFRs are addressed by the M2M transformation and others remain under the final responsibility of the developer. To sum up, we may state that MDD approaches that are not able to deal with NFRs in the software production process suffer from severe drawbacks that must be manually fixed by either the developer or the MDD engineer and that, therefore, may compromise their adoption.

The situation is even worse when considering not the theory but the real state of practice of MDD, hampered by the limitations of MDD tools available in the market. For instance, their code-generation capabilities are limited to particular technologies/languages (which implies that only some parts of the system can be transformed and generated by the tool), and it is not always easy to change the predefined M2M and M2T transformations offered by the tool. Therefore, a scenario more realistic than those depicted in Figure 3.1 is described below (see Figure 3.2):

1. The MDD engineer specifies a PIM that contains only information about system functional aspects.

2. The software architect defines the transformations that are applied to different parts of the PIM, generating each a part of the target PSM. Each generated PSM part is compliant with a particular technology.

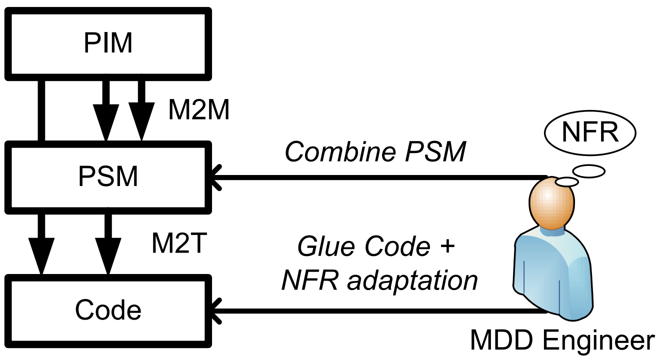


Figure 3.2: Dealing with NFRs using current MDD technologies

3. M2T transformations are applied to the PSM for obtaining the final code.
4. The developer complements the generated code and combines the generated code excerpts into a coherent architecture.

This process is adding some new drawbacks:

- There is not a single transformation generating a complete PSM, but a set of partial transformations generating separated pieces that may yield an incomplete PSM. Even, some tools skip the generation of the PSM and jump directly to the code.
- The different pieces generated by the transformations need to be manually linked, writing additional glue code.
- With respect to NFRs, each transformation results on PSM parts that may not satisfy the stated NFRs (in fact, depending on the available transformations each excerpt can enforce different and maybe contradictory NFRs).

It is worth to mention that, even having such difficulties, it has been corroborated in a survey that *"there are more respondents for whom code* *generation has a positive impact on their productivity than there are those for whom the integration of generated code is a problem"* [115].

3.2 Motivation

In this section we present an academic exemplar that we will use in the rest of the paper for illustration purposes.

The ACME travel agency offers transportation and accommodation services. The management has decided to deploy a web portal in order to offer some online functionalities to its customers, e.g., user management, payment facilities and searches (hotels, flights, etc.).

Together with these functionalities, many NFRs appear during the requirements elicitation process. For example, since the portal is providing e-commerce transactions, security requirements like R1 = "The system shall detect and report unauthorized data accesses" are a must. The effect of this NFR can be manifold, for instance in a Web-based environment, firewalls are an architectural solution that supports this goal.

Other NFRs depend on the specific characteristics of the travel agency and the planned portal usage. Let's consider two scenarios:

Scenario 1. ACME is a specialized travel agency that offers luxury vacation packages to exotic destinations in 5-star hotels. It has a reduced portfolio of clients that plan their vacations using the system.

Scenario 2. ACME is a world-wide leader travel agency. The company offers hundreds of packages that are assembled by combining data imported from other transportation and accommodation sites.

These scenarios impose some NFRs that capture their most essential characteristics. Thus, in Scenario 1, the number of expected visits is not too high and therefore scalability is not an issue. On the contrary, scalability and availability are key concerns to ensure the success of the portal in Scenario 2. Clearly, a good production process should be sensible to these differences and should result in different systems for each scenario. To make this statement more evident, let's consider one particular system dimension, the deployment architectural view as defined by Kruchten [141].

Table 3.1: Effects of components on some architectural properties

	SSC	DBMS separated	DBMS and AS separated	Replication
Performance	Poor	Average	Good	Improve
Scalability	Poor	Poor	Poor	Improve
Availability	Poor	Poor	Poor	Improve
Maintenance	Good	Average	Average	Damage
Security	Poor	Good	Good	Neutral
Complexity	Good	Average	Poor	Damage

The deployment architectural view refers to the physical distribution of the software system components. Since the system we are considering as exemplar is a Web application, we may identify the following types of components [58]: the Web Server (WS), the Application Server (AS) and the Data Base Management System (DBMS). All these components can be deployed on the same node (Single Server Configuration, SSC), or using one of the several possible separations of components (e.g., separation of the DBMS). Also in the design of the deployment architecture it is possible to consider any type of component replication. Each deployment strategy affects some software quality attributes [67]. For instance, component replication (e.g., WS and AS) supports scalability, because more simultaneous connections may be established; replication also may improve efficiency especially if a load balancing component coordinates the incoming traffic. Table 3.1 summarizes these strategies on some types of NFRs, according to [58].

At this point, the software architect has the duty of choosing the most adequate deployment strategy for the given set of NFRs, by comparing them with the effect of each strategy on the quality attributes. For the two scenarios described above, examples of convenient options are:

Scenario 1. The DBMS is kept separated from the WS and AS since scalability and availability are not major concerns, whilst security is increased by placing a firewall between the DBMS and the other two components (see Figure 3.3-a). Replication is not implemented since its benefits are again concerning criteria that are not important for the given NFRs, whilst others would be damaged.

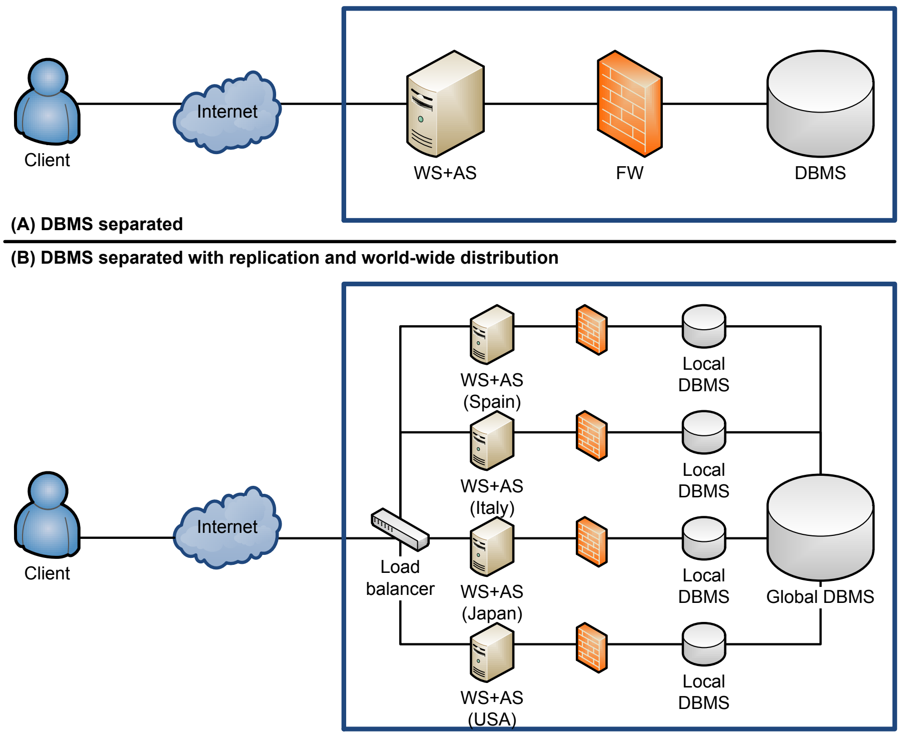


Figure 3.3: Two different deployment architectures for the Web portal case

Scenario 2. Since the agency provides a world-wide service, the WS and AS are replicated to improve availability and performance in those sites for which a greater number of clients may be expected. A load balancing system coordinates the different WS to improve performance even more. DBMS containing data local to the sites are put together with the WS and AS, and firewalls are also deployed for protecting each local DBMS. As a final decision, a centralized DBMS contains some replicated data that may be of interest for performing some data mining operations. Figure 3.3-b, provides the whole picture.

Other deployment options are possible. It is not a goal of this section to discuss them, but just to emphasize the fact that the final form of the software architecture depends on the set of elicited NFRs and to give some initial idea of the type of knowledge to manage and decisions to be made.

Using the actual MDD methods, for example, Executable UML [154], the travel agency model consists of use case diagrams, class diagrams, sequence diagrams and activity diagrams that express the roles, functionalities, data and behavior of the system. None of these artifacts is able to express any kind of NFR. Thus, the transformation from PIM to PSM is fixed and it is not possible to choose the most appropriate strategy for a given set of NFRs: the PSM will be close to, or far from, the elicited NFRs depending on the system quality factors implicitly encoded in the predefined transformations.

3.3 Introducing NFRs in MDD

In the Section 3.1 we have shown that MDD approaches that do not consider NFRs as part of the generation process suffer from serious drawbacks, and that, unfortunately, this is the predominant type of approach nowadays. In this section we discuss a general solution to this problem.

As we have seen in Chapter 1, many authors have reported the intimate relationship among requirements and architectures and also the great impact that NFR have on architectures [164, 63, 103]. For example, in Section 3.2, we have shown how new components (e.g., firewalls and load balancers) and physical component allocation (e.g., replication) can be justified in terms of the NFRs that must be satisfied. Therefore, we envisage an approach to MDD in which the PIM is transformed into a complete software architecture. Transformations have the mission of allocating the responsibilities coming from the PIM functional part to components that are deployed into an architecture that satisfies the NFRs.

But NFRs are also important when determining the choice of technologies needed to implement the architecture. For instance, it may be necessary not just to know that a relational data base is needed, but also that a particular brand, or even version and release, is the right choice. Interoperability requirements (e.g., "The portal shall be compatible with our current data base in the central management system") or non-technical requirements [56] (e.g., "The data base vendor shall provide 24×7 call center assistance") are clear examples of NFRs with this effect.

Table 3.2 describes the main elements that constitute the envisioned framework proposal. Remarkably, and following the discussion above, we introduce two kinds of models between the PIM and the code: the model representing the architecture, and the model representing the technological solution. Whilst the latter is clearly a PSM, the former lays in between the two levels of abstraction and therefore we denote it by PIM/PSM. For each kind of model, we include between parentheses the requirements that are satisfied by the elements in that model. Finally, as a consequence of having two different intermediate models among the PIM and the code, we have two corresponding M2M transformations, M2M_{arch} and M2M_{tech}.

3.3.1 NFR-aware MDD: NFRs in the PIM

We argue that the most natural way to integrate NFRs into the MDD process is by considering NFRs from the very beginning of the development process, i.e., as part of the PIM. As functional requirements, NFRs become first-order citizens of the MDD process.

The MDD process then works as follows (see Figure 3.5):

- The analyst specifies a PIM that contains both the functional and non-functional requirements, PIM(f+nf).

- The MDD decisional engine decides, given the PIM(f+nf) and the contents of the MDD knowledge base (i.e., information about non-functionality, architectures and technologies), the final form of the transformation M2M_{arch}:

  $$\mathit{M2M}_{\mathit{arch}}\text{: } \mathit{PIM(}\text{f+nf}\mathit{)} \rightarrow \mathit{PIM/PSM(}\text{f+nf}_0\mathit{)}$$

  This transformation takes PIM(f+nf) as input and produces as output PIM/PSM(f+nf_0), a model describing an architecture that implements all the functionality f in a way that satisfies the elicited subset of NFRs nf_0 whose satisfaction depends on the decisions made at the architectural level.

- Once the PIM/PSM(f+nf_0) has been generated, the MDD decisional engine applies a second M2M transformation that generates the PSM for the desired final implementation technology. This PSM follows the architectural guidelines expressed above (and thus, satisfies nf_0) but

Table 3.2: Concepts needed when integrating NFRs into MDD

Concept	Definition	Example
f, nf	The elicited functionality and non-functionality of the system (not represented as model)	An IEEE 830-compliant Software Requirements Document
PIM(f)	PIM that specifies some functionality *f* of the system	A UML class diagram specifying the system data
PIM(f+nf)	PIM that specifies all the requirements of the system	An *i** model of the system complemented with UML data and behavioral diagrams
PIM/PSM(f+nf)	Model mixing PIM and PSM levels that specifies some functionality *f* satisfying the NFRs *nf*	A 3-layer architecture expressed with the ACME Architectural Description Language (ADL)
PSM(f+nf)	PSM that specifies some functionality *f* satisfying the NFRs *nf*	A model with a class diagram annotated with database stereotypes (e.g., «PrimaryKey», «Table») that only have meaning for the Oracle DBMS
Code(f+nf)	Executable system that implements the functionality *f* satisfying the NFRs *nf*	Implementation of the 3-layer architecture above using Java components, XML interchange data formats, Oracle DB schema, etc.
M2M	M2M transformation from a PIM to a PSM	Transformation of a UML specification into a technological solution including an Oracle data base and a Pound load balancer, among others
$\mathrm{M2M}_{\mathrm{arch}}$	M2M transformation from a PIM to a PIM/PSM that represents the architecture of the system	A mapping from an Executable UML model of functionality into a 3-layer architecture expressed with the ACME ADL
$\mathrm{M2M}_{\mathrm{tech}}$	M2M transformation from a PIM/PSM into a PSM that represents the technological solution of the system	Transformation of the ACME architectural model into a representation of technology that, for instance, annotates a class diagram with Oracle-compliant database stereotypes
M2T	M2T transformation from a PSM to the executable system	Transformation of a stereotyped UML diagram to EJB Java classes

also takes into account all the remaining *nf* (directly related to technologies), forcing the adoption of a particular technology or product:

$$M2M_{tech}\text{: } PIM/PSM(\mathrm{f{+}nf_0}) \rightarrow PSM(\mathrm{f{+}nf})$$

- Last, a simple M2T transformation can be applied to obtain the code from the technology:

$$\mathit{M2T{:}\ PSM(}\mathrm{f{+}nf}\mathit{)} \rightarrow \mathit{Code(}\mathrm{f{+}nf}\mathit{)}$$

In the framework, the transformations are presented as single functions. In fact, this is a simplified view since a transformation will be in fact a composition of the application of many transformation rules. Thus, we may say (being M2M either $\mathrm{M2M}_{\mathrm{arch}}$ or $\mathrm{M2M}_{\mathrm{tech}}$) that:

$$\mathit{M2M(m){:}}\ \mathrm{rk}_{m2m}(\ldots(r1_{m2m}(m)\ldots)$$

From a conceptual point of view, the vision of the transformation as a single function is a convenient simplification that does not hamper the generality of the approach.

Example: deciding the need of firewall components

In this example we illustrate the kind of information to record, and steps to apply, in order to derive part of the architectural model for the Web portal example presented in Section 3.2. We remark that the notations used to represent the models, and even the concrete steps taken and their order are just an example of how they may look like, we refer to Section 3.4 for further discussion.

We distinguish three parts: the knowledge base used by the MDD decisional engine; the creation of the starting PIM; and the application of our MDD process itself. For the latest, we will restrict to the creation of the PIM/PSM.

1. Representing the MDD knowledge.

   We focus on the concepts directly related to NFRs. First, it is necessary to represent the types of NFRs managed and the consequences that architectural decisions may have on them. We can represent this using a tabular structure (like Table 3.1, page 35) or by means of a notation like the NFR framework [63], used with similar purposes in several works (e.g., [101, 37]). The model depicted in Figure 3.4 top-left, shows an excerpt of the information needed, with several softgoals to represent the NFRs and two particular operationalizations for them (each with a different positive/negative effect on them).

Next, it is necessary to represent the implications of each operationalization on the architecture. This is described for the firewall case in the bottom-left part of Figure 3.4. The firewall solution requires three participants: the firewall component itself, and two subsystems that are connected, the internal (i.e., protected) and the external ones. These elements are in fact instances of architectural metaelements, e.g., subsystem, defined according to some architectural description like those in [141, 76].

2. Creating the PIM(f+nf).

   The process starts with the PIM definition. For the functional part we can still follow any existing proposal, e.g., Executable UML. For the NFRs, we may decide to use a natural language representation based on requirement patterns as in [182]. which allows to establish easily the link between such NFRs and the predefined NFRs types in the MDD knowledge base (KB). For instance, Figure 3.4 top-right represents the R1 NFR (see Section 3.2) and the link with the Security NFR type maintained in the KB.

3. Creating the PIM/PSM.

   The MDD decisional engine chooses, using some appropriate analysis technique (e.g., [63, 112]), the Firewall operationalization to support R1.

   As a consequence of the system being a Web application (which is a decision coming from the intrinsic nature of a Web portal), a transformational step decomposes the system into three main subsystems: WS, AS and DBMS. The MDD Knowledge Base knows that the communication between these subsystems is: WS $\leftrightarrow$ AS $\leftrightarrow$ DBMS.

   The assignment of elements from the functional part of PIM(f+nf) into WS, AS and DBMS, takes place. In particular, the data model elements are assigned into DBMS.

   Since R1 is referring to data protection, and since DBMS is bound to data, the MDD decisional engine decides that the protected subsystem for the firewall is the DBMS. Since the communication for Web application is from AS to DBMS, it is also possible to deduce that the *source* of the Firewall is the AS.

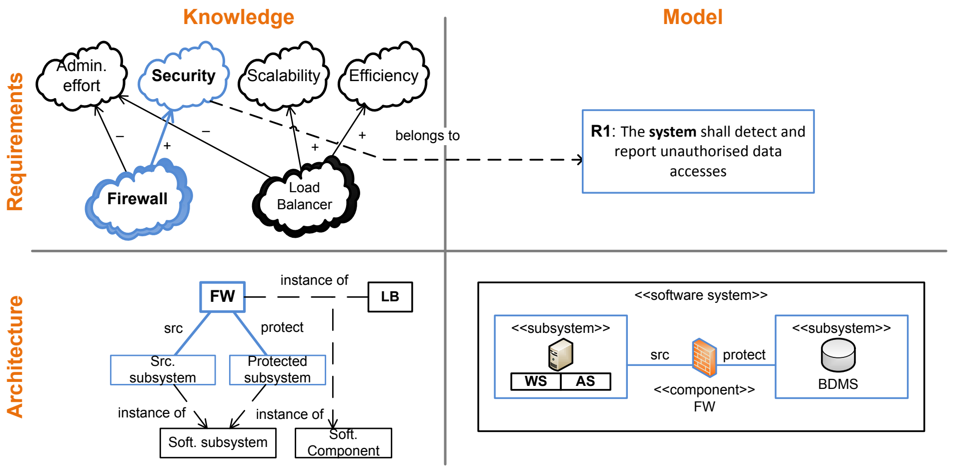


Figure 3.4: Knowledge and models used in the example

In Scenario 1 (see Section 3.2), since there is no replication, there are just one AS and one DBMS, and thus just one Firewall is induced (see Figure 3.4 bottom-right). In Scenario 2, due to replication, there are as many Firewalls as pairs AS-DBMS. The fact that the WS and the AS are deployed together completes the information needed to determine the final form of the architecture.

3.3.2 NFR-aware MDD: NFRs for decision-making

Although the framework presented above is theoretically neat, it is clear that it has a lot of complexity to manage. Remarkable, it requires:

- To determine the most adequate formalism for representing the non-functional part of PIM($f+nf$). We have used in the example the NFR framework, that is basically a qualitative-oriented one, but also more quantitative approaches may be considered, e.g., in QoS-style [187].
- To embody in the MDD decisional engine all the knowledge needed to make informed architectural decisions, i.e., to determine the concrete form that the M2M functions take. In other words, the M2M are

required to provide a correct output in all possible situations. This is a very strong condition mainly because of: first, the amount of knowledge to represent is huge and not always clear; ans second, the conflicting nature of NFRs, i.e., architectural decisions permanently require trade-off analysis.

These problems lead to propose a second alternative. Instead of considering NFRs as part of the PIM and then be an input of the MDD process, we may consider that the MDD process asks the software architect the NFR-related information as it is needed. The resulting process becomes:

- The analyst specifies a PIM that contains just the functional requirements, PIM(*f*).
- The transformation function $\mathrm{M2M}_{\mathrm{arch}}$ takes PIM(*f*) as input and produces PIM/PSM including *f* and nf_0 (nf_0 stands again for those NFRs that concern the architecture). To produce this output, the MDD process presents a series of questions Q = $\{q_1, ..., q_n\}$ to the software architect whose answer is needed in order to decide the transformation steps to apply. The software architect provides answers A = $\{a_1, ..., a_n\}$ according to the NFRs nf_0. If we denote by σ_{arch} the function that records the mapping from questions to answers, $\sigma_{\mathrm{arch}}(q_i)=a_i$, the transformation function is defined as:

  $$M2M_{arch}\text{: } PIM(\mathrm{f}) \times \sigma_{arch} \rightarrow PIM/PSM(\mathrm{f+nf_0})$$

- The subsequent M2M transformation for the technology acts the same, requiring a similar σ_{tech} function to obtain from the MDD engineer the information needed to make informed decisions:

  $$M2M_{tech}\text{: } PIM/PSM(\mathrm{f+nf_0}) \times \sigma_{tech} \rightarrow PSM(\mathrm{f+nf})$$

- The M2T transformation is not affected:

  $$M2T\text{: } PSM(\mathrm{f+nf}) \rightarrow Code(\mathrm{f+nf})$$

Questions that the MDD decisional engine may raise to the architect may be manifold. For instance, there may be high-level questions like the type of organization with respect to departments (e.g., to decide which nodes are

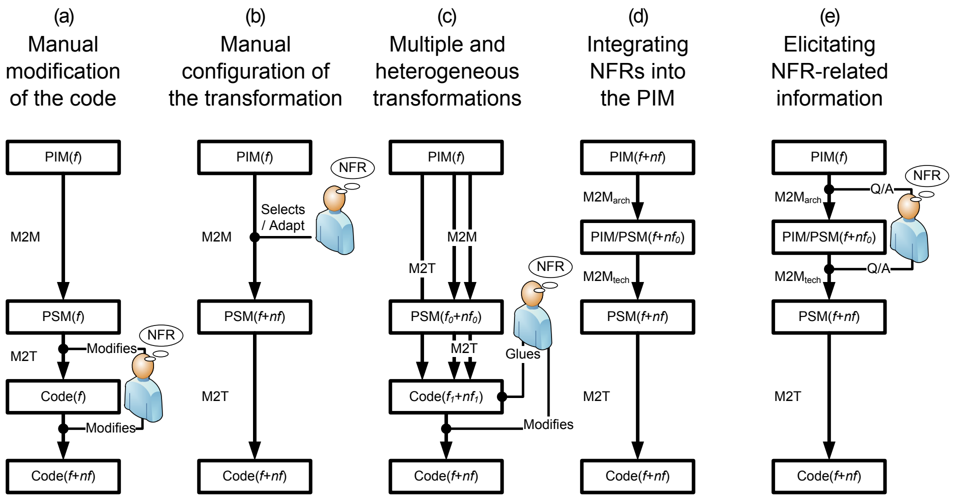


Figure 3.5: Comparison among the different MDD strategies analyzed

part of the physical architecture) and lower-level ones like the probability of execution of a given operation or use case.

The two NFR-aware approaches presented in this section represent two extreme visions but of course we can think of hybrid solutions, in which the MDD decisional engine supports decision-making for some types of NFRs, architectural elements and technologies, whilst the software architect and developer may provide the information missing under demand.

This second alternative is specially interesting while clearly accepted technical solutions for the mentioned points in the first alternative are not provided.

3.3.3 Comparison

In this section we compare the two NFR-aware approaches presented in this section with the three approaches presented in Section 3.1. Figure 3.5 aligns the five approaches studied in this chapter; *a*, *b*, and *c* represent the state of practice, and *d* and *e* the approaches proposed in this work (whose details were explained in sections: 3.1.1, 3.1.2, 3.1.3, 3.3.1, and 3.3.2 respectively).

In Figure 3.5, it holds that $nf_0 \subseteq nf_1 \subseteq nf$. When comparing, please pay attention to: the number and nature of models and transformations;

the extent of requirements in the models (enclosed in parenthesis); and the type of interaction with the human assistant.

Table 3.3 includes a detailed comparison respect to criteria that includes aspects of the project setup, the production process, and the product management. As summary, the main benefits of the NFR-aware approaches presented here are:

- NFR-aware approaches fully integrate NFRs into the MDD process. Especially in the first NFR-aware framework presented (Figure 3.5-d), NFRs are considered at the same level than the functional specification, being both part of the departing PIM. Knowledge may be incrementally stored in the MDD knowledge base (gradually improving accuracy of results) and may be reused in each new project.

- As a consequence, there is no need for the developer neither to write glue code (since the different components of the PSM model are already interrelated) nor to adapt the code to satisfy the NFRs (since the NFRs have been already taken into account when creating the PSM model).

- Instead of obtaining several incomplete PSM, using a single transformation that targets a specific architecture a single, a comprehensive and unified representation of the system is derived.

- Two levels of abstraction are recognized, one for representing architectures, other for representing technologies. This distinction fits with the levels of abstraction that practitioners use in their daily work.

- The explicit representation of NFRs allows defining model transformation repositories inside the MDD knowledge base that can be used to select the proper transformations to apply. Also, when NFRs are considered at the PIM level, classical analysis techniques from Requirements Engineering may be applied in the early stages of the MDD process.

- Hybrid approaches (between options from Figure 3.5-d and 3.5-e) allow customizing the NFR-awareness to the preferences of software architects. An empirical study that we conducted shown that software architects are reluctant to lose control over the architectural decisions [21, 20] (empirical results are presented in Chapter 5).

Table 3.3: Comparison among the different MDD strategies analyzed

Project set-up	(a)	(b)	(c)	(d)	(e)
Modeling time	Fair (just functionality is modeled)	Fair (just functionality is modeled)	High (several notations used to build the PIM)	Very high (NFRs need to be modeled)	Fair (just functionality is modeled)
MDD configuration time for a particular project	None (transformation applied as is)	Probably high (if a new transformation is needed)	None (transformations applied as are)	None (transformations applied as are)	None (transformations applied as are)
Production process	(a)	(b)	(c)	(d)	(e)
Production time once configuration finished	High (full post-process adaptation)	Fair (slight post-process adaptation will probably be needed)	Very high (post-process adaptation and gluing)	None (if transformations are complete)	Low (guided conversation with human)
Criticality of human intervention during production	High (high responsibility of the developer at the end)	Fair (slight post-process adaptation will probably be needed)	High (high responsibility of the developer at the end)	None (since there are no human interactions)	Low (she just needs to respond to very concrete questions)
Complexity of the process	Low (the MDD infrastructure is static)	High (several transformations co-exist)	Very high (several heterogeneous trans-formations exist)	High (the transformations used will be more complex)	Moderate (human intervention simplifies the process)
Knowledge reuse and learning ability	Very low (just the functional-related knowledge is reused)	Low (learning ability comes from the MDD engineer)	Very low (just the functional-related knowledge is reused)	Very high (NFR-related knowledge may be reused and may grow)	High (some NFR-related knowledge may be reused and may grow)
MDD KB maintenance cost	Low (since it just covers functionality)	Fair (updates up to the MDD engineer)	Fair (updates up to the MDD engineer)	Very high (all new knowledge needs to be modeled)	High (some new knowledge needs to be modeled)
Product management	(a)	(b)	(c)	(d)	(e)
Product Traceability	Very low (generated product modified)	Fair (depending on the complexity of the post-process adaptation)	Extremely low (generated product modified; information across models)	Potentially complete (all decisions can be traced)	High (answers to questions may be recorded)
Maintainability	Very low (changes made are probably lost if product generated again)	Fair (depending on the complexity of the post-process adaptation)	Very low (changes made are probably lost if product generated again)	Very high (it is possible to work only at PIM level)	High (functionality at PIM level; changes on NFRs require new questions)

But as Table 3.3 shows, these benefits are not for free. Incorporating NFRs results in higher modeling effort, both for constructing the PIM and for building the MDD knowledge base. Also, it requires discipline to keep this MDD knowledge base up to date. Complexity of the MDD process is the overall challenge to overcome.

3.4 Discussion

Although some NFR-aware development processes have been formulated in the past, putting a new NFR-aware MDD production process into practice looks like a great challenge. In this section we outlined the most relevant issues to investigate with emphasis on requirements-related issues.

Modelling of NFRs at the PIM-level

- Which types of NFRs are most relevant to the MDD process? It is important to devote efforts to the NFRs that software architects perceive as the most important. Surveys (e.g., [21, 20]) and interviews are needed.
- Which notation to use for representing NFRs? As commented, quantitative and qualitative approaches are the two (non-exclusive) big families. This is an old research topic in Requirements Engineering (already appearing in the 2000's roadmap [160]) and results obtained in contexts other than MDD may be transferred here.
- How NFRs may be linked to the functional elements? Some approaches have been cited [88, 169, 170] at this respect.

Elicitation and representation of architectural knowledge

- Which types of architectural knowledge exist and how are they used in practice? Again empirical studies are needed to give light to this question [68].
- Which are the quality attributes corresponding to these styles?
- Which are the matching rules that allow determining the architectural solution that best fits the required NFRs?

Nature of models

The classification of MDD models into CIM, PIM and PSM as defined in the MDA approach results in some rigidity in our context. We have already defined the architectural model as an intermediate PIM/PSM model. The situation may even be more confusing if we inject the concept of architectural view [141] into the core of the MDD process. For instance, we may envisage that the evolution from the PIM down to the architectural models yields to a sequence of models in decreasing abstraction level, corresponding to the different architectural views, from the logical view down to the physical view. In this case, labeling the models may be artificial. We remark too that current MDD approaches focus on the architectural logical view, thus addressing other views is a progress by itself.

M2M transformations

Challenges are:

- Gradually developing and incorporating in the framework transformations for all popular architectural styles.

- Selecting the best alternative for each non-deterministic transformation depending on the expected NFRs.

- Defining a transformation composition language for gluing separate transformations into the MDD models. This last point is highly connected with the vision promoted in [188, 186] where different types of NFR are handled separately. Being true that the particularities of each NFR type makes it difficult to treat them uniformly, it is also clear that we need to be able to reconcile them since the generated system needs to fulfill all of them together.

- The framework presented here conceives the application of transformation (and thus generation of models) top-down with respect to abstraction level. However, this does not need to be always this way. For instance, a technological NFR fixing the brand and release of the data base product will have an implicit consequence on some other more abstract model, namely to know that a data base of a particular type (relational, OOR, etc.) has to be integrated into e.g., the development view of the architecture.

The MDD core: decisional engine and knowledge base

The research agenda includes:

- Being able to keep and reuse the knowledge acquired in MDD projects (e.g., success and failure experiences).
- Exploring the applicability of artificial intelligence techniques for taking informed decisions (case-based reasoning, Bayesian networks, etc.).
- Exploit the knowledge of software architects to improve the automation of the process by means of a comprehensive program of interviews and surveys.
- Define the roles and responsibilities that play a part in the MDD process: software architect, MDD engineer, software developer, domain expert, etc.

Variations from the proposed frameworks

Being the presented frameworks general, variations may be thought to be formulated. Let's consider one particular variation, namely the incorporation into the MDD process of the concept of architectural style. According to [196, 51], an architectural style consists of the description of the types of architectural components supported, their organization, and how can they be integrated. In some sense, we may say that different architectural styles use different ontologies, e.g., whilst SOA talks about services, choreography and MOM, layered architectures introduce layers, persistence and push communication model. Incorporating this concept into the framework has consequences on its very core. If the M2M translation from PIM to PIM/PSM renders a software architecture, it follows that each architectural style requires a different metamodel, thus both PIM/PSM models and M2M transformations are dependent on the architectural style, becoming families of models and functions:

$$\left(M2M_{arch[st]}\text{: } PIM(\mathrm{f+nf}) \rightarrow PIM/PSM_{[st]}\ (\mathrm{f+nf}_0) \right)_{st \,\in\, style}$$

Determining the architectural style should be the first decision to be made in the MDD process. Adopting a pure MDD perspective, it should be determined from the PIM(f+nf). However, it is true that the decision of whether it must be, for example, an SOA or a Web rich client architecture is often a decision made before the MDD process starts for reasons that are not always tangible and are only in the architect's mind.

Correctness and completeness issues

Last but not least, we mention the need of accurately investigating the notion of correctness of an NFR-aware approach. We may envisage the following conditions that need to be refined to the chosen formalisms. A couple of examples of predicates to investigate are:

- The NFRs should be correct both independently (e.g., there are not contradictory NFRs) and when referred to the functionality f (each functional element is qualified by meaningful types of NFRs): correct(nf) $\wedge$ applicable(nf, f)
- The knowledge embedded in the MDD knowledge base should find feasible alternatives for any given NFRs that fulfil the correctness and applicability conditions above: correct(nf) $\wedge$ applicable(nf, f) $\Rightarrow$ reducible(KB, nf)

3.5 Conclusions

In this section we have: explored the state of the art; envisaged some generic solution to the identified problems; and enumerated new lines of research and challenges to overcome; of one requirement-related practice, the management of non-functional requirements (NFR) in the model-driven development (MDD) production paradigm.

The main goal has been to agree on a perspective of the current state of the addressed problem and in the need to keep progressing towards several directions. Concerning the state of the art:

- We have analyzed how MDD methods not dealing with NFRs behave to ensure their satisfaction.
- We have run a systematic literature review to learn insights of the MDD methods that deal with NFRs.

Concerning the improvement of this state of the art:

- We have outlined an NFR-aware general framework which allows customization to different settings with their own peculiarities
- We have discussed variations on this framework.
- We have aligned and thoroughly compared the different alternatives discovered, trying to make clear not just the benefits but also the obstacles of this general framework.

All in all, we agree with the observation in [108]: "[...] MDD has a chance to succeed in the realm of large, distributed, industrial software development, but it is far from a sure bet". We hope to contribute to boost the MDD adoption by practitioners and the design of more powerful MDD methods and better software production processes, and thus increases the likelihood of this bet.

Part II

NFRs in Software Architecture

Chapter 4

State of the art

This chapter is based on the related work presented in [11, 25].

As we have seen in the introduction, NFRs express desired qualities of the system to be developed. They refer both to observable qualities such as system performance, availability and dependability, and also to internal characteristics concerning, e.g., maintainability and portability. Other authors use different names, remarkably *quality requirement*, as a synonymous of NFR, being the diversity of terminology and meaning well-known by the community [98].

Over the years, a common claim made by software engineers is that it is not feasible to produce a software system that meets stakeholders' needs without taking NFRs into account. As a result, software development projects currently invest a lot into satisfying NFRs [49]. But still it seems to be a lopsided emphasis in the functionality of the system, even though the functionality is not useful or usable when NFRs do not hold [62].

The tight relationship among NFRs and software architectures (SAs) is part of the established body of knowledge in software engineering. As early as in 1994, Kazman and Bass made the point that asserting that SA is intimately connected to the achievement of NFRs should not be controversial [127]. This vision has pervaded over the years and explains why software projects invest a lot into fulfilling NFRs [49]. This influence is mentioned recurrently in the literature: NFRs often influence the system architecture more than functional requirements do [179]; *"the rationale behind each architecture decision is mostly about achieving certain NFRs"* [217]; *"business goals and their associated quality attribute requirements strongly influence a system's architecture"* [173].

This general statement can be made more concrete if we consider the evolution of the concept of SA from a simple structural representation to a decision-centric viewpoint [144]. Under this perspective, *"[NFRs] play a critical role during system development, serving as selection criteria for choosing among myriads of alternative designs and ultimate implementations"* [62]. For example, deciding a layered architectural style may be justified in terms of maintainability or portability, or choosing a particular technology can be motivated by an efficiency gain.

The previous mentioned works provide little direct evidence from real case studies to support these statements. Both requirements engineers [43] and software architects [86, 6] demand field work to sustain or dismiss that *"much of a software architect's life is spent designing software systems to meet a set of quality attribute requirements"* [100].

4.1 Empirical studies on NFRs

In spite of their acknowledged importance, not so many empirical studies centered on NFRs are available. A recent systematic literature review conducted by Svensson et al. [43] found no more than 18 empirical research studies centered on investigating the benefits and limitations on methods around NFRs for five identified areas: elicitation, dependencies, level of quantification, cost estimation, and prioritization. The need to increase the number and quality of studies on NFRs was pointed out as a key finding of the review.

- Svensson et al. have also conducted several empirical studies on the topic. In [41, 42], they focused on the analysis of practices on companies that produce market-driven embedded systems. Svensson et al. targeted several aspects on NFRs in [41], whilst in [42] they focused on issues related to requirements prioritization. The findings of this last paper suggest that there seems to be a lack of knowledge about managing NFRs in these companies; the authors hypothesize that this could be related to the lower importance given to them with respect to functional requirements (this is a recurrent argument in several studies). [41] reports a different perception of some NFRs aspects depending on the role of the interviewee (e.g., project managers ranked performance as the most important quality aspect, whilst project leaders ranked usability first), which supports the idea of replicating empirical studies for the different roles that participate in software development.

- Borg et al. studied in depth two case studies in two Swedish companies [46]. They interviewed 7 professionals for each case. They reported some common findings in both companies (e.g., vagueness of NFRs and difficulty to test), but also some differences, remarkably in the provenance of requirements, which was different in both cases due to contextual factors. The main conclusion of their study is that although both organizations were aware of the importance of NFRs, still their main focus was on functional requirements. The authors made the hypothesis that methods and tools supporting NFRs throughout the entire development process would be the best way to fight against this situation.

- De la Vara et al. [79], presented an e-survey with 31 valid responses. It was conducted with the purpose of analyzing the importance of the different types of NFRs depending on factors like type and size of project, role of the observer and application domain. Concerning role, they checked that the same three types were identified by the three analyzed roles, although the importance of the types could vary.

- Haigh [107] analyzed the importance of 13 types of NFRs with an e-survey of 318 responses, the participants were recently graduated students. In the paper Haigh tries to determine if there are differences between the priorities given to the types of NFRs and the types of participant (users, managers, or developers), and of there are differences between the priorities given to the types of NFRs and the types of software, which include (enterprise administration, office package, development tool, process control, business analysis, scientific and technical software, and other). As result, some differences were found, for example usability is more important for users, while accuracy is for managers, and maintainability is for developers.

- Anh et al. [26], explored several issue related to OSS adoption projects. One of the research questions was about the degree of satisfaction of NFRs by selected OSS components. The authors explored different types of NFRs and showed that performance and reliability are the two types considered most important by interviewees, and that this last type is the worst fulfilled by the components.

- Daneva et al. [74] presented an exploratory study about how architects deal with quality requirements in the context of large and contract-based software projects. Many of the research questions of this work, such as how are the requirements elicited, documented, and validated, have a match with the research questions presented in the second empirical study [11] (see Chapter 5). Some of the answers to these research questions differ between the two studies. The most likely explanation, is that these differences have the origin in the nature of the studied projects, ours were small and medium scale, while in [74] were large projects.

4.1.1 Analysis

Table 4.1 shows a summary of the analyzed studies. Compared to these empirical studies on NFRs, the main value of ours (presented in Chapter 5) is the focus on the relation between NFRs and the software architect role. In none of the previous studies this relationship was the real subject of study and thus available evidence is anecdotal, which makes our own study appealing, especially considering the claims that the software architect role is one of the most affected by NFRs. We believe that our studies bring some new interesting observations to the field.

4.2 Empirical studies on software architecture

As mentioned in the beginning of this chapter, there is a tight relation between NFRs and software architectures. This relationship is not supported by empirical studies in the software architecture area.

- Tang et al.'s work on architecture design rationale [201] provides evidence that our subject of research is highly relevant for software archi-

Table 4.1: Summary of empirical studies on NFRs

Ref.	Subject of research	Type of study	Companies	Population
[41]	NFR importance NFR dependencies NFR satisfaction	Interviews	5 companies	5 project leaders 5 product managers
[42]	NFR prioritization	Interviews	11 companies	11 project leaders 11 product managers
[46]	NFR elicitation NFR documentation	Interviews	2 companies	14 (different roles)
[79]	NFR importance	e-survey	25 companies	6 product managers 14 project leaders 11 programmers
[107]	NFR importance	e-survey	Not specified	162 users 110 managers 46 developers
[26]	NFRs in OSS	Questionnaire	15 companies	15 (different roles)
[74]	NFR elicitation NFR documentation NFR importance NFR prioritization	Interviews	14 companies	20 architects

tects. This paper discusses the role of the architect in comparison to the dedication to different tasks and the design of NFRs appears third in the list (of interest for 64.2% of interviewees), right after overall system design (86.4%) and requirements or tender analysis (81.5%). However, the paper does not further discuss the relationship of SA and NFRs.

- Ali Babar et al. [4] reported observations about documentation and validation of software architectures. Participants declared that having a good understanding of the types and levels of required quality attributes is a vital factor as the types of attributes to be evaluated usually have significant influence on the choice of methods and practices.

- Heesch and Avgeriou [208] drove a descriptive survey. From the results, they propose some best practices, for example: the possibility that architects get involved in the requirements elicitation for a better understanding of the requirements and other architectural drivers.

Table 4.2: Summary of empirical studies on software architecture

Ref.	Subject of research	Type of study	Companies	Population
[201]	Architecture design	Questionnaire	Not specified	81 software architects
[4]	Architecture design, documentation and validation	Group discussion	10 companies	10 software architects
[208]	Architects reasoning	e-survey	Not specified	53 software architects
[38]	Service-Oriented Architecture	e-survey	Not specified	29 respondents
[181]	Architects and NFR	e-survey	Not specified	39 architects

- Becha and Amyot [38] drove an electronic survey to identify the non-functional properties relevant to software oriented architectures. Some of the results are: the need of separation of responsibilities between service providers, consumers and network providers and the need of a standardized vocabulary for the non-functional properties.

- Poort et al. [181] presented the analysis and key findings of a survey about dealing with non-functional requirements (NFRs) among architects. They find that, as long as the architect is aware of the importance of NFRs, they do not adversely affect project success, with one exception: highly business critical modifiability tends to be detrimental to project success, even when the architect is aware of it.

4.2.1 Analysis

Table 4.2 shows a summary of the analyzed studies related to software architecture. Since available empirical studies are not many, having new ones that provide further evidence in topics already explored may also be considered valuable.

Chapter 5

State of the practice

Relation between the contents of this chapter and published papers.

Section 5.1 Main contributions of [20, 21].

Section 5.2 Main contributions of [11, 12].

Section 5.3 Main contributions of [25].

This chapter includes three empirical studies (see Table 5.1):

- *First empirical study.* This empirical study was designed and executed in 2008, and the results were published in 2009 with partial results [20] and in 2010 [21]. The study is based on an electronic survey that obtained 60 answers.

- *Second empirical study.* This empirical study was designed in 2009, executed in 2009-2010, and the results were published in 2012 [11, 50, 12]. The study is based on 13 personal interviews to software architects.

- *Third empirical study.* This empirical study was designed in 2011, executed from May to September of the same year, and the results were published in 2013 [25]. This study is based in an electronic survey that obtained 31 answers.

Table 5.1: Summary of performed empirical studies

Study	Subject of research	Type of study	Companies	Population
1st	NFRs in practice	e-survey	Not specified	60 software architects
2nd	NFRs in architecture	Interviews	12 companies	13 software architects
3rd	NFRs in SBSs	e-survey	Not specified	31 software architects

This studies go from a wide scope to narrow scope. The first was a general overview of the industrial practice, the second was oriented only to architects, and the third one, to the particular case of Service-Based Systems (SBSs). The three studies have in common the focus on the role of NFRs in architects practice.

5.1 First empirical study

To know about the current industrial practice of software architects and the role of NFRs we drove an electronic survey addressed to software architects and we obtained 60 responses that give some light to questions such as: how do architects consider NFRs, and what are the most influential types of NFRs in their daily work.

5.1.1 Design

The survey was designed in an iterative way. In each iteration, it has been revised by IT experts and/or researchers of the area. There have been three iterations, the first one in plain text and the other two using the final electronic format. We chose the LimeSurvey[1] tool to produce the electronic version of the questionnaire.

Research Questions

This survey was oriented to obtain answers from the current industrial practice. The research questions of this study are listed in Table 5.2.

[1]`http://www.limesurvey.org`

Table 5.2: Research questions of the first empirical study

ID	Research Question
RQ1	How important are NFRs to practitioners?
RQ2	Is the NFRs importance dependent on the architectural style used?
RQ3	Are NFRs well integrated in the current development tools?
RQ4	What is the desired level of interaction with semi-automatic tools?
RQ5	What is the current state of adoption of MDD techniques by the industry?

- RQ1: We wanted to determine how are NFRs perceived in the industry, so the first research question is: *How important are NFRs to practitioners?*

- RQ2: We also were interested in knowing if the importance of NFR is dependent on the architectural style used, so we defined the next research question: *Is the NFRs importance dependent on the architectural style used?*

- RQ3: Since we already were thinking in developing a tool to make architectural decisions from NFRs, we wanted to understand the practitioners' needs for development tools. The third research question is: *Are NFRs well integrated in the current development tools?*

- RQ4: One part particular aspect that concerned us was what practitioners expect from these kind of tools, the research question is: *What is the desired level of interaction with semi-automatic tools?*

- RQ5: Finally, since our intention was to develop a tool for decision-making upon the MDD framework proposed in Chapter 3, we wanted to know how is MDD perceived by the practitioners. The fifth research question for this study is: *What is the current state of adoption of MDD techniques by the industry?*

Questionnaire

The complete questionnaire is available in Appendix A. It is divided in three parts:

- *First part.* Questions about software development. Concretely, we asked about the used architectural styles, the type of developed applications, the platforms and technologies used, and questions about Non-Functional Requirements (NFRs). In this part we obtained contextual information and the answer to the RQ1 and RQ2.
- *Second part.* This part had two questions about the desirable interaction of a hypothetical development tool. The first question is about code generation and the second is about design decisions. With this questions we will find an answer to RQ3 and RQ4.
- *Third part.* In this part we asked to the participants questions about their use of MDD techniques to answer RQ5.

Data Collection

For the dissemination of the survey we used two strategies. On the one hand, personal contact with 10 software architects that we know personally, and on the second hand, advertisement in IT communities hosted in web sites such as LinkedIn (e.g., the International Association of Software Architects group). The survey was available online during 2009, from March to October (8 months).

Data analysis

To ensure the quality of the data obtained from the questionnaire, we applied sanity checks to find obvious errors in data. We used descriptive statistics to analyze the data [132].

Limitations of the study

We have analyzed the construct validity and the internal and external validity of this study.

Construct validity. This aspect of validity reflects to what extent the operational measures really represent what is investigated according to the research questions [213]. This study was supported by two main principles: rigorous planning of the study according to Oates [166], and establishment of protocols for data collection and data analysis. The survey was piloted both internally and using external participants from the target population.

Internal validity. There might have been confounding variables and other sources that could bias our results [65, 135]. To control variables, exclusion or randomization can be applied [208]. Exclusion means that participants who are not sufficiently experienced were excluded from the study. We ensured this by having a question about the participants that use NFRs to make architectural decisions. Randomization means that we used a sampling technique which leads to random participants. Furthermore, validity is subject to ambiguous and poorly phrased questions. To mitigate this risk, we piloted the data collection instrument in multiple iterations until potential respondents understood our questions and intentions. Also, participants might not have had the same understanding as we required them (e.g., what is a NFR, what is a architectural style); we tried to mitigate this risk by including the definitions of the concepts that were susceptible of being hard to understand by the target population.

External validity. External validity is concerned with the problem of generalizing the results to the target population. We assume that our results are applicable to a population that meets the sampling criteria of our survey (i.e., software developers).

Population

We obtained 60 responses (it is worth to remark that more of than the 50.0% were from Spain). The participants defined themselves majorly as software developers project managers, and architects, but there were also some specialists in some software development technology (see Figure 5.1).

One of the questions that raised during this study was: when do we have enough amount of answers? We did not reach a satisfactory answers to this question, because we cannot know how big is the population of software architects in the world, we also cannot know the amount of people that were invited (because of the second type of dissemination strategy). Finally we accepted 60 responses as enough because other empirical studies in software architecture had similar amount of answers.

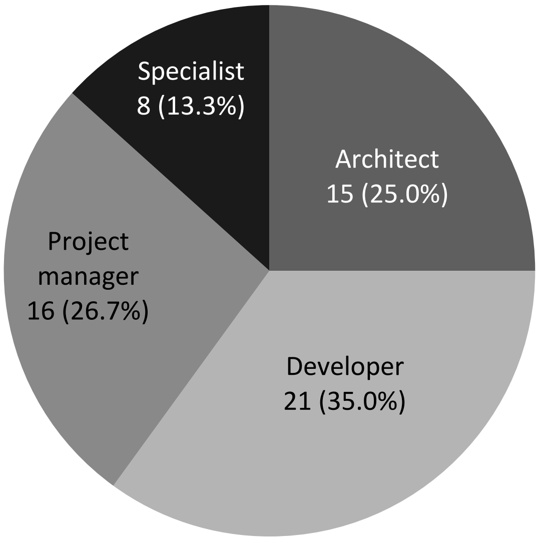


Figure 5.1: Population of the first empirical study

5.1.2 Results

RQ1: How important are NFRs to practitioners?

The first questions where about the role of NFRs in their habitual software development practices:

- 96.0% of respondents consider NFRs in their development practices.
- 73.0% said that NFRs are at the same level as functional requirements.
- 57.0% said that NFRs are used to make architectural decisions.

Respondents rated (in a Likert scale) nine types of NFRs which comprises five of the six quality characteristics of the ISO/IEC 9126 [117] (functionality was not included), and the other four were suggested by the testers of the survey: reusability and three non-technical types (standard compliance, cost, and organizational). The respondents rated these types of NFRs with respect to the importance that they have in their projects (see Figure 5.2). In this figure we can see that reliability is the only type of NFR considered critical by a majority of respondents. While organizational and portability types of NFRs have a median of *medium* importance, the rest of NFRs have a median of *important* importance. It was a bit surprising for us that the cost requirements and organizational requirements were of lower importance (in comparison to others). This situation may be dependent on the participants role in the software development (e.g., business role vs. technical role), but if we only consider the participants with *project management* role the results indicate that they are more worried about reliability with a median of *critical* importance.

Answer: NFRs are clearly important for practitioners, even if only 73.0% of respondents considered them as important as functional requirements, and, more surprisingly, only 57.0% used them to make architectural decisions (here is worth to remember that not all the survey participants were architects). In fact, almost all the types of NFRs analyzed were considered important by the respondents (i.e, its median is *important*).

RQ2: Is the NFRs importance dependent on the architectural style used?

The same analysis was repeated, but in this case limiting the considered answers (A) to the respondents that used NFRs to make architectural decisions, and these were grouped in the three architectural styles studied[2]:

- 3-Layers architecture ($A = 26$ answers). The 3-Layers architecture shows that all types of NFRs with the exception of portability, have a median of *important* importance.
- Service-Oriented Architecture (SOA) ($A = 20$ answers). In the case of MVC, the reliability has a median of *critical* importance, and only portability has a median lower than *important* importance.
- Model-View-Controller (MVC) architecture ($A = 24$ answers). In the case of MVC, reliability has a median of *critical* importance.

[2]Note that some respondents answered to develop using more than one type of architectural style.

It is worth to mention that limiting the answers in this way, all the respondents considered that at least the importance of any requirement is of *marginal* importance, having no respondent saying that a requirement is not important at all. A general observation is that the respondents that use NFRs to make architectural decisions gave higher rates to all the types of NFRs, which is an indicator that they are more concerned with NFRs.

Answer: it seems that the importance of NFRs is not hardly conditioned by the use of a particular architectural style, but the respondents that work with these architectural styles are more concerned with NFRs in general.

RQ3: Are NFRs well integrated in the current development tools?

In this survey we asked about the use of tools in software development, the first question was about a tool to help in the analysis of NFRs compliance. We obtained that 80.0% of respondents declared that the development tools that they use are not well-suited for analyzing the compliance of the specified NFRs, whilst 70.0% would like to have them.

Answer: NFRs are not well integrated in the current development tools, and practitioners would like to have support for them.

RQ4: What is the desired level of interaction with semi-automatic tools?

We also asked questions about the amount of interaction desired with an hypothetical support tool for implementation and design tasks. We considered the following tasks:

A. Generation of the skeleton code.

B. Generation of the code for a specific technology.

C. Selection of the architectural style using NFRs.

D. Selection of the technological style using NFRs.

Architectural style and technological style were defined in the survey as follows:

Architectural style We understand architectural style as the collection of the main elements that compose the software system and the strategy of communication used between them. Examples of software architectures are: 3-layered architecture, service oriented architecture, client-server, etc. A software system can be designed as a composition of many architectural styles depending on its needs.

Technological style A technological style is a set of technologies to construct the elements that compose the software system. A technological style must consider all necessary technological roles of the implementation: platform, programming languages, libraries, technological standards and external services (e.g., database management systems or authentication services). The technologies that takes part in a technological style must be able to work jointly.

The respondents had to choose one of the following interaction levels for each task:

1. I would not use any support tool to perform this task.

2. The support tool should ask me before taking any decision.

3. The support tool should ask me only before taking the relevant decisions.

4. The support tool should take the decisions for me but later I would check them.

5. The support tool would take the decisions for me without further confirmation.

The results obtained in these questions are presented in Table 5.3 (rows represent analyzed tasks and columns the interaction level).

Table 5.3: Desired interaction levels

	1	2	3	4	5
A.	8.3%	15.0%	23.3%	41.7%	11.7%
B.	8.3%	20.0%	26.7%	41.7%	3.3%
C.	13.8%	13.8%	50.0%	20.7%	1.7%
D.	12.1%	20.7%	39.7%	25.9%	1.7%

Answer: in general practitioners want to have the last word in the architectural decisions made, this means that a completely automatic tool won't suit their needs. It is also evident that practitioners are more inclined to trust a tool for implementation tasks rather than design tasks.

RQ5: What is the current state of adoption of MDD techniques by the industry?

Finally, we asked about the use of MDD tools, a great part of the respondents (78.3%) said that they do not use MDD in their daily projects, but 51.7% of them declare to know what is MDD. Eclipse (with a 35.0%) seems to be the most known platform for MDD, and MDA (with 48.3%) the most known initiative.

Answer: MDD techniques are far from being adopted by the industry. Even if the techniques are known, practitioners are still reluctant to use them.

5.1.3 Discussion

After analyzing the previous results we had some observations.

- One observation is that, surprisingly, only the 57.0% said to use NFRs to make architectural decisions, this result does not support what is said in the literature about the importance of NFRs in architecture (see Section 1.1.2).

- Another striking observation is that while 80.0% are not able to check their NFRs, the 70.0% would like to have such kind of tool. For us this is a clear indicator that there is an unsatisfied need in software industry.
- About the use of tools, we found an indicator that architects want to have the last word in decision-making. This finding has driven some of the design decisions of the method presented in Section 7.2.

After performing this survey, we decided that it was necessary to obtain more empirical evidence of the current state of practice, in particular for the software architecture area.

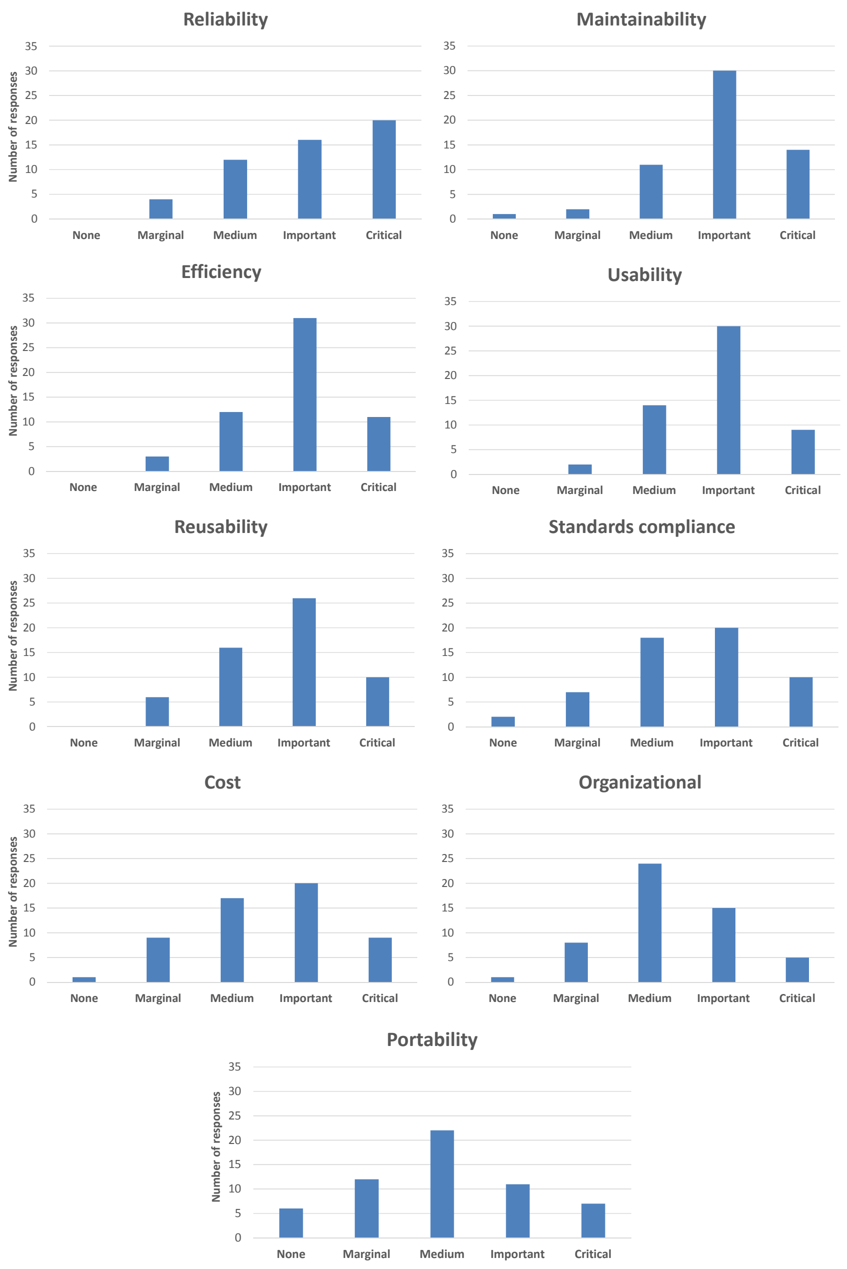


Figure 5.2: Importance of NFRs by types

5.2 Second empirical study

In this section we present an empirical study that uncovers some relevant software architects' practices in dealing with NFRs. The study is implemented as an exploratory survey and was conducted over our local network of software architects. Based on the analysis of the answers, we were able to draw some observations about the use and impact of NFRs in industrial practice, align them with the results of previous empirical studies, and discuss possible actions that could eventually help to improve the state of practice in the field.

5.2.1 Design

We carried out an exploratory study using a qualitative research [156]. Qualitative research is especially indicated when the purpose is to explore the subject of interest with the aim of improving the knowledge available. The general goal of investigating how software architects deal with NFRs was decomposed into several research questions shown in Table 5.4. Although the focus is on NFR-related issues, we added a preliminary research question about the responsibilities that software architects have assigned in their organizations to help understanding and interpreting the results. The other research questions focused on the perspective of the software architect on elicitation, documentation, validation and tool support, as well as terminology issues, the importance of NFR types, and the influence of NFRs on architectural decision-making.

We used semi-structured interviews for gathering information about a single software development project in which the respondents participated as software architects. Compared to other qualitative research strategies (e.g., structured questionnaires) semi-structured interviews provide more flexibility and allow investigating in more depth interesting issues that appear during the conversation. On the other hand, considering a single project instead of a general perception of the architects' rationale allows for better interpretation and assessment of contextual information [70]. The interview guide was carefully designed following the guidelines stated by Oates [166]. The interview guide used in the study is available in the Appendix B.

Table 5.4: Research questions of the second empirical study

ID	Research Question
RQ1	What is the role of the software architect?
RQ2	Are there terminological confusions on NFRs?
RQ3	What types of NFRs are relevant to software architects?
RQ4	How architects deal with NFRs?
- RQ4.1	Who elicits the NFRs?
- RQ4.2	How are NFRs elicited?
- RQ4.3	How are NFRs documented?
- RQ4.4	How are NFRs validated?
RQ5	What type of tool support for NFRs is used?
RQ6	How do NFRs influence architectural decisions?
- RQ6.1	What types of decisions are driven by NFRs?
- RQ6.2	How is the architectural decision-making process?
- RQ6.3	How NFRs influence architectural decisions?

Population

The target population of the study (see Table 5.5) were professionals that covered the role of architect in at least one project in the organization. Under McBride's perspective [153], a software architect is the person who makes design and technological decisions in a software development project. It is important to remark, though, that we did not provide this or any other definition to interviewees, on the contrary RQ1 was precisely intended to find out the view that they had on software architect's responsibilities.

Participating organizations were chosen from our industrial collaboration network. We sent invitation letters to 21 software-intensive organizations and asked for their willingness to participate in the study. We finally recruited 12 organizations covering a varied spectrum of business areas and application domains. The organizations were of three different types:

- *SCC:* software consultancy companies that perform software development tasks for different clients as its primary business.

- *ITD:* IT departments in public or tertiary organizations that usually perform or outsource some software development tasks for covering the internal demands of the organization
- *SH:* software houses that develop and commercialize specific proprietary solutions.

In one of the organizations, we were able to interview two software architects, bringing the total number of interviews to 13. The respondents held different positions in the organizations and were in charge of architectural tasks in at least the project they based their answers on. Most respondents had an education background related to computer science (with just two cases of academic background related to telecommunications and industrial engineering). 11 of the respondents had a bachelor's degree. The projects themselves were also diverse in terms of functionality and size. Although all organizations were from Spain, some of the projects clients were from abroad.

Execution

The interview guide was sent to the respondents in advance; therefore they became familiar with the topic and were able to choose the project beforehand. Interviews were conducted face-to-face in the respondents' mother tongue. Each interview took about one hour and was recorded and prepared for analysis through the manual transcription (made by an external company) of the audio records into text documents. Once transcribed, documents were validated by the respondents. In a few cases, they were explicitly requested to clarify some aspect that remained uncertain after all the interviews were completed.

Analysis

Data analysis was conducted in a series of steps (based on [156]).

1. The interview transcripts and individual notes taken during the interviews were coded independently by two researchers. This codification consisted in identifying and classifying relevant parts of the text transcribed.

Table 5.5: Overview of the organizations

ID	Area[1]	Main Domain	Project Description	Staff	Roles played[2]	Dur.[3]
A	SCC	Domain-specific IS (lottery)	Web application for managing transactions over mobile phones	15	Arch, Dev	6
B	ITD	Domain-specific IS (University school)	Web application for activity management	3	Arch, PM	48
C	SCC	General-purpose IS	System for the management and logistics of a growing fast-food chain	5	Arch	120
D	SCC	Aerospace IS	Geographic information system to manage aerospace launch bases	≈10	Arch	180
E	SCC	General-purpose IS	Application to manage the processes and documents in public-sector	6	Arch, Dev	30
F	SCC	Web IS	E-commerce system for selling motorcycle items	5	Arch, Dev, PM	12
G	SCC	Geographic IS	Web system to support shipping logistics	1	Arch, Dev, PM	3
H	SCC	Web IS	Web system for personal data management	≈20	Arch, Dev	36
I	SCC	Document management IS	System to manage bank accounting activities	8	Arch, Dev	18
J	SH	Domain-specific IS (insurance)	System to support insurance company tasks	50	Arch, Dev	30
K	ITD	Domain-specific IS (University school)	System to manage staff research activities	8	Arch, PM	36
L1	ITD	Domain-specific IS (University dpt.)	Web to manage students activities	5	Arch, Dev, PM	144
L2	ITD	Domain-specific IS (University dpt.)	Web collaboration system	8	Arch, Dev, PM	5

[1] SCC: Software Consultancy Company; ITD: IT department; SH: Software house.
[2] Arch: Architect; Dev: Developer; PM: Project manager.
[3] Project duration expressed in months.

2. We used the tabulation technique [156] to analyze the answers of each question of the interview guide. This made it possible to get an overview of the responses and ease the process of categories generation. Depending on the granularity of the questions, some of them got a higher number of categories. We used the NVivo Software[3] to support this process.

3. Once the answers were processed, we compared the results. Most of the categories generated by the two researchers were semantically similar, but some others needed further discussion.

4. All the researchers of this study met for several discussion meetings to generate categories by grouping sentences or phrases that described the same idea, action, or property. Whenever we had a disagreement, we discussed the issues until we reached an agreement. As a result, some categories were split, modified, discarded or added to ensure that all answers were well-represented.

5. Finally, for displaying the results shown in this paper, we used the counting technique [185] to enable the reader to *see* the findings by counting frequency of occurrences, or recurrent categories of events.

The results are presented in Section 5.2.2.

Limitations of the study

Like all other empirical studies, this one also faces certain validity threats. This section discusses them in terms of construct, internal and external validity as well as reliability, as proposed by Yin [213] and also emphasizes the mitigation actions used.

Construct validity. This aspect of validity reflects to what extent the operational measures really represent what is investigated according to the research questions [213]. This study was supported by two main principles: rigorous planning of the study according to Oates [166], and establishment of protocols for data collection and data analysis. Our protocol included

[3]`www.qsrinternational.com`

specific mitigation actions for evaluation apprehension by ensuring the confidentiality of the interviews and also by emphasizing the exploratory nature of the study. In addition, the interview guide used as an instrument to gather data, was piloted with 2 academic and 2 industrial people in order to improve its understandability. As a result, some changes were done to enhance the elicitation process (e.g., we added a glossary to homogenize key terms that could cause some confusion).

Internal validity. It refers to the confidence that we can place in the cause and effect relationship in a study [213]. We took relevant decisions for approaching a further understanding of the approached research questions. One of the main relevant decisions was to focus the questions of the interview guide on a single software development project. Considering a single project instead of a general perception of the architects' rationale allows for better interpretation and assessment of contextual information [185]. It would otherwise have been very difficult to interpret certain decisions or influential factors related to the nature of the projects. We are aware that some possible biases may be related to this strategy, for instance the fact that some time passed since the project was completed, so it could be difficult for the respondents to remember some project details. To reduce the possible side effects of this, we sent the interview guide in advance to the respondents so they could become familiar with the topic, and asked them to choose the project beforehand. Thus, when performing the study, we rarely experienced respondents having difficulty in remembering project details. Another factor raised was that the projects were selected by the participants. They may have selected the most successful project to base their answers on, although we asked them to use the most familiar one. To mitigate this, we explained that our study was not focused on analyzing *best practices* but on learning *how things are done.* There is always the possibility that the respondent forgets something or does not explicitly state it when s/he is asked about it [193]. To reduce this issue, we approached two strategies:

- We discussed some potential topics that might be omitted by the respondents, and paid particular attention to them during the interviews in order to ask for clarifications if necessary.
- Once the interviews were transcribed, the documents sent back to the respondents for validation.

We tried to be rigorous with respect to the data analysis strategy, and put forward several mitigation strategies:

- To a better understanding and assessment of the data gathered we recorded all interviews (and later on transcribed them).
- To reduce the potential researcher bias, two different researchers assessed the data individually and generated their own categories.
- The generated categories were analyzed, discussed and reviewed by all researchers of the study to ensure their accuracy, understanding and agreement.
- The categories were checked with respect to the data gathered to confirm that none of the categories refuted any of the conclusions.

External validity. It is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case [213]. As our study was exploratory, we do not attempt to make universal generalizations. Thus our observations should be interpreted not as a universal view of the field status but as a starting point for a universal discussion and analysis [190].

Moreover, we did not randomly select the organizations that participated in the study but got them from our industrial collaboration network. However, we tried to strengthen the external validity by having no control over the projects chosen by the respondents. It is important to mention that most of the participating companies were small or medium-sized, in addition, most of the studied projects dealt with non-critical domains (except for aerospace and banking). We are aware that both factors may have an impact on how NFRs are dealt with, and so we highlight that our findings should be considered with caution.

Reliability. This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. In order to strengthen this aspect we considered the validity of the study from the very beginning. So, as stated in the previous paragraphs, we put forward several strategies. In addition, we maintained a detailed protocol, the collected data and obtained results were reviewed by the participants; we have spent sufficient time with the study, and gave sufficient concern to the analysis of all responses.

5.2.2 Results

We present next the most relevant observations resulting from the analysis of the interviewees' responses. We include quotations from the interviews stating respondents' ID in bold and enclosed in parenthesis. For each research question, we start describing the findings, followed by a summary, then our opinion and relations to results from other studies, and finally a short answer for the research question.

RQ1: What is the role of the software architect?

The analysis of the interviewees' responses in our study shows that at their companies, the role of architect did not exist as a job position as such. Given this situation, we tried to understand more in depth how architects were nominated for this role and how the boundaries of this role were set.

How were software architects nominated? The nomination of the respondents as software architects was made according to the nature of the project. In other words, it was not based on the usual architects' skills defined in the literature [153], but on technical knowledge (*"[The architect] is whoever knows the technologies used in the development best"* **(E)**) or experience (*"Decisions affecting the whole system are made by the most experienced people"* **(H)**).

How was their role scoped? Respondents found it difficult to define the exact nature of their work as an architect since it overlapped with other activities they performed in the project, primarily, project management (7 respondents) and development (9). Some even played two other roles apart from architect. Only one respondent said that the only role he played in the project was that of architect.

Summary of findings

- 13 interviewees performed the tasks assigned to "software architects" in the project based on their experience or knowledge rather than their possible skills as architects
- 0 interviewees held a "software architect" position at the company
- 12 interviewees played other roles in the project in addition to the role of software architect.

Opinion: Most software engineering literature concurs that software companies have a specific position, known as the "software architect," whose mission is to design an architectural solution in a software development project by making architectural decisions that are compliant with the elicited requirements. Some authors as McBride [153] and Clements [66] support this statement. However, our results show that the role of architect did not exist as a job position in the organizations and that their tasks were very diverse. This also concurs with other studies not specifically reporting on the software architect role but related to RE, e.g., Sadraei et al. reporting on project managers to take on RE activities [192].

On the one hand, the respondents were nominated as architects of the assessed projects mainly based on their technical knowledge. This finding aligns with the stated opinions of other professionals, e.g., *"an architect should only be responsible for a single project/application and not the architect for all projects within a software company"*[4].

On the other hand, it was difficult to enumerate the architect' tasks as these overlapped other roles' tasks. This fact aligns with the observation made by Tang et al. [201] who state that architects work on a variety of tasks (such as requirements analysis, tender analysis, architecture design and software design) and management responsibilities.

Answer: software architects did not exist as a differentiated role and performed other duties in the projects.

RQ2: Are there terminological confusions on NFRs?

In our interviews we encountered certain communication problems concerning the meanings of words, especially with regard to the definition of types of NFRs. In fact, we found two related problems: the problem of meaning itself, and the problem of translating English terms into another language, Spanish in our case[5]. We had to handle this aspect carefully during the interviews and later data analysis. Four interviewees **(E, F, G, I)** required additional clarifications when being interviewed (e.g., about the meaning

[4]`joncahill.zero41.com/2009/04/role-of-software-architect.html`

[5]We use English terms for the discussion but the problems appeared in their Spanish use.

of “availability” and “accuracy”), two **(B, E)** used an inappropriate term in a given context (e.g., “ergonomic” meaning “usable”), and one used a term with an incorrect definition (e.g., *“Maintainability is very important, because when something is working, we can’t make changes”* **(D)**. Another recurrent matter was the indiscriminate use of the terms “Performance” and “Efficiency” that required further questions from our side. At this respect, it is worth to mention that the standard ISO/IEC 25000 [116] is proposing the term “Performance efficiency” defined as *“performance relative to the amount of resources used under stated conditions”*, which can help to clarify this confusion.

Summary of findings

- Confusion was reported around the terminology for designating NFR types

Opinion: The problem of gathering data from interviewees was challenging due to the terminological discrepancies and misunderstandings about concepts related to NFRs. It is not the practitioners the (only) ones to blame, their confusion just reflected the lack of consensus that exist in the community, e.g., in the use of “performance” and “efficiency”.

This problem has been also highlighted by other researchers. E.g., Anh et al. [26] reported confusion among maintainability and reliability (*“OSS components are more reliable because the code is available and then it is easier to fix the bug”*). Also, Svensson et al. reported that the concept of “compliance” as used by some interviewees was fairly different from the ISO/IEC 25000 standard’s [116], e.g., some respondent said that compliance is important because *“we must be compliant with the requirement document”* [41]. Last, the problem of using English terminology by non-English professionals was reported also in [46] where the majority of practitioners was native Swedish speakers and had troubles when documenting the requirements in English.

Answer: architects did not share a common vocabulary for types of NFRs and showed some misunderstandings.

RQ3: What types of NFRs are relevant to software architects?

We asked the respondents what were the NFR types that they took into account when making architectural decisions. We consolidated their answers, e.g., to reconcile different names for the same concept (see the terminology problem above) using the ISO/IEC 25000 standard [116] as unifying framework. Some respondents had problems to directly answer the question. In those cases, we provided them with a list of 15 terms that was consolidated when piloting the survey design and clarified their meaning. Figure 5.3 shows the result of this part of the interview.

If we observe the bar chart, we may see a graduation of the mentioned types. This aligns with the information given by the architects that considered some types as common sense characteristics, e.g., *"I consider Performance and Security as default requirements of any project"* **(B)**, *"I would never think on a system that it is not Secure"* **(I)**. Apart from these dominant types, we found other situations:

- NFRs that were considered because they represented an explicit need of the client, e.g., *"one of the contractual requirements was that the system could interoperate with other systems that were already deployed in the client's environment"* **(D)**.
- NFRs that were particularly important for the development team, e.g., *"We were the ones that would maintain the system, so, it was important for us to ensure its maintainability"* **(B)**.
- Four of the respondents mentioned that some NFRs were not important to them because they rely on the technologies and the underling platform, e.g., *"We didn't thought about the security of the documents because it is done by the management system of SharePoint"* **(E)**. The perception was that the maturity level of many technological solutions was enough to ensure the satisfaction of NFRs.

Moreover, about 40% of the NFRs considered by respondents in their projects were non-technical [56], i.e., referring to issues not directly related to the quality of the product itself but to some contextual information. In fact, some respondents explicitly mentioned that some types of non-technical NFRs took precedence over all others (*"Money rules and everything has to be*

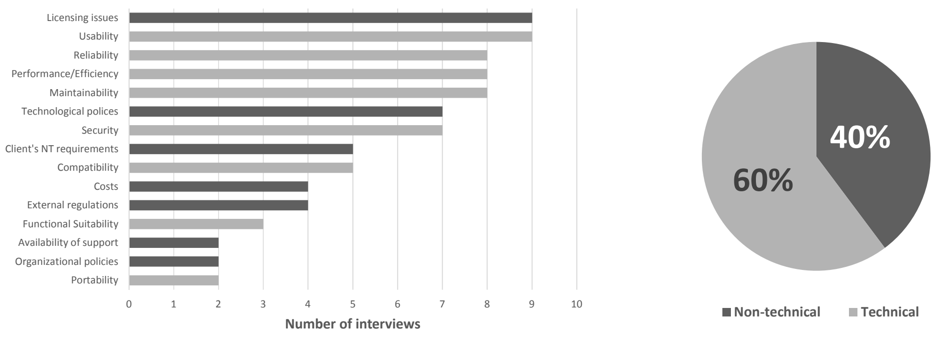


Figure 5.3: Importance of NFR types (technical and non-technical)

adapted to it" **(J)**). The types of non-technical NFRs most often mentioned were (see Figure 5.3):

- Licensing issues, 9 times (e.g., *"the client's organization limited the type of OSS licenses to be used in the software solution"* **(J)**)
- Technological constraints, 7 (e.g., *"we prefer to use technologies we have already mastered"* **(L1)**, *"we had some limitations from the client, for example, architecture based on OSS and Java"* **(H)**)
- Client's NT requirements, 5 (e.g., *"we needed to adapt our solution to the organization's strategic vision"* **(I)**)
- Cost, 4 (e.g., *"we preferred JBoss to an IBM solution because of cost constraints"* **(F)**)
- External regulations, 4 (e.g., *"as we are a public organization, we had to comply with certain public regulations and make our system accessible for people with certain disabilities"* **(L1)**)
- Availability of support, 2 (e.g., *"the choice of technology was influenced by the support that Oracle offered"* **(A)**)
- Organizational policies, 2 (e.g., *"we preferred to use our own human resources instead of subcontracting someone else"* **(J)**)

Summary of findings

- 49 references were made to technical NFRs (60%)
- 33 references were made to non-technical NFRs (40%)

Opinion: If there is a topic that has been documented in existing empirical studies with respect our research questions, is the perception of the importance of NFR types. However, since these studies did not focus on the software architect role (with the exception of the first empirical study presented in this thesis, Section 5.1), it was a good opportunity to complement these findings with our observations.

The higher importance of performance and usability was also reported in two previous studies, [41] and [79], in the last case together with maintainability. In [26] performance was important, but usability was not among the most important quality attributes. It is worth to mention that it is not easy to align the results of these studies since often they use different classification for NFRs, therefore we have not tried to make an exhaustive alignment of the results.

In spite of these similarities, we have observed too that the results are still dependent on the development team, experience, and domain (e.g., aerospace domain **(D)**, gave much importance to safety of people, whilst this type of NFR was not mentioned by the other participants). These facts could be a factor influencing the partially divergent results with previous studies. Similar opinions appear in other empirical works (e.g., *"[NFRs] importance can vary depending on stakeholders' roles, types of project, orders of magnitude of requirements and application domains"* [79]; *"NFR types that are typical for traditional telecommunication systems gain more attention than others"* [46]). Also, in [43] and [79] it is mentioned that the role of the stakeholders may influence on the perception of importance for the NFRs, but this difference could not be observed in this study because all our participants played the architect role. On the contrary, we found one work stating that there are NFR types that are always important, e.g., *"some quality requirements (security) are always important for everyone"* [42]. Similar statements were made by the architects interviewed in our study (e.g., *"I would never think on a system that it is not Secure"* **(I)**), but it is worth to mention that security was also mentioned as example of NFR type whose satisfaction is delegated onto the technologies used (from the architects perspective), and in consequence not considered important.

In our study we found out that non-technical NFRs are considered by the architects almost as important as technical NFRs. As far as we know, no other empirical study made this differentiation, even though that some of the non-technical NFRs are recurrently mentioned (e.g., cost [201]).

Answer: the two most important types of technical NFRs for architects were performance and usability. And architects considered non-technical NFRs to be as relevant as technical NFRs.

RQ4.1: Who elicits the NFRs?

Our interviews show that in 10 out of the 13 projects considered, the software architect (the interviewee) was the main source of the NFRs. Clients either never mentioned NFRs (*"[the client] never mentioned that web pages could not take more than 2 seconds to load, but he complained about it afterwards"* **(E)**) or provided only very broad indications, usually in the form of cost or efficiency constraints (*"the client mentioned a basic [NFR], and we added others based on our experience"* **(L2)**). The main explanation seems to be that architects consider themselves to be the real experts when it comes to defining efficiency, reliability, and other similar aspects.

Respondents **(D)**, **(H)**, and **(I)** were the only three cases with client-led NFR elicitation process. Interestingly, they were also the only cases in the study in which the interviewee was working on an outsourced project (managed by an aerospace company **(D)**, a software company **(H)**, and a bank **(I)**). Even in these cases, however, the architects played an active role in completing the definition of the NFRs (*"Our client was an aerospace system department. Therefore, all the NFRs were very well defined. We also added other NFRs based on our experience"* **(D)**).

Summary of findings

- In 10 projects, the NFRs were elicited solely by the architect
- In 3 projects, the NFRs were elicited by the client with the participation of the architect

Opinion: Requirements elicitation is the process of obtaining the system requirements from stakeholders and other sources and to refine them in greater detail. It is considered by both researchers and practitioners to be one of the most challenging activities in the requirements engineering process. Numerous techniques (interviews, role playing, etc.) have been developed for requirements elicitation in a precise and unambiguous manner. They usually assume that the client, as the domain expert, is the main source of requirements. In fact, some respondent acknowledged that when referring to functional requirements: *"[Business analyst] writes a detailed document reflecting all the [functional] requirements specified by the customer"* **(A)**.

However, when it comes to NFRs, our interviews show that this assumption did not hold: in 10 of the 13 projects considered, the software architect was the main source of the NFRs. This number exceeds the already high rate (60%) reported by van Heesch and Avgeriou on architects involved either completely or a lot, in requirements elicitation [208]. From the kind of answers they gave, we conjecture that architects consider themselves to be the real experts (at least, more than customers) when it comes to defining efficiency, reliability, and other similar aspects (*"the same way I do not recommend my customers how to implementing the accounting procedures, I do not expect them to tell me how to organize my architecture"* **(C)**). There is another empirical work covering this aspect, Borg et al. [46] reported from two studied cases that in one it is said that requirements elicited directly from end users are very rare, whilst in the other, most of the requirements are elicited directly from customers and end users.

Answer: NFRs were mainly elicited by the architects themselves.

RQ4.2: How are NFRs elicited?

The elusive nature of NFRs can make them difficult to elicit upfront. According to this general belief, all respondents agreed that deciding NFRs is a gradual and iterative process throughout the system life-cycle. The reason is that part of the expected behavior of the system may not be uncovered until a certain milestone has been achieved *"We determined first some relevant NFRs (e.g., compatibility with other systems) and then developed a prototype and analyzed alternatives"* **(J)**.

Furthermore, the interviewees emphasized that the list of NFRs of the project could never be considered complete even after the development tasks had finished, instead, this list is under extension and negotiation during all development and maintenance phases of the project, e.g., *"In relation to efficiency we had to make changes because the necessary level of service was not specified at the beginning of the project"* **(K)**. There is a clear justification of this situation. Contrary to functionality, some NFRs such as those related to security cannot be completely checked until the system is deployed in its expected environment and some unexpected conditions arise.

Summary of findings

- 13 architects considered elicitation as a gradual process

Opinion: The iterative nature of NFR elicitation has not been explicitly stated by other studies. Some weakly related statement may be found by Doerr et al., who argue that the elicitation of NFRs, functional requirements and the architecture must be intertwined [83], which seems to imply that NFRs cannot be elicited upfront. Also the finding stated by Svensson et al. in [41] about NFR dismissal is somehow related: a total average mean 22.5% of NFRs were reported to be dismissed whilst the projects evolved.

Answer: NFRs were mainly elicited following an iterative process.

RQ4.3: How are NFRs documented?

Academics and standards organizations have proposed throughout the years many notations and templates to write system requirement specifications. They are supposed to facilitate the requirement documentation activity making it more efficient. However, our study shows that 9 out of the 13 interviewees acknowledged that they had not documented the NFRs at all (*"[functional requirements] came in UML, using conceptual models and use cases, but there was no mention of NFRs"* **(H)**). Some interviewees emphasized that documentation is only necessary if the client or the critical nature of the domain requires it.

The respondents who explicitly documented their NFRs used different methods to do so:

- Volere templates [184] **(B)**.
- Grouping of the NFRs using the ISO/IEC 9126 quality classification [117] **(K)**.
- Domain-Specific Language (*"Since we work in the field of aerospace, our NFRs had to be clearly stated and verifiable. We have special templates, and we used different techniques from other engineering disciplines, such as risk models, failure trees, etc."* **(D)**).
- Simply wrote a plain text document **(J)**.

Out of these four, two (**(J)** and **(K)**) only documented the initial NFRs (*"At first, we wrote down some initial ideas for NFRs in natural language [...], but afterwards we did not keep track of any of them or of any other NFRs arising during the design process"* **(K)**).

Summary of findings

- 9 architects did not document the NFRs at all
- 4 architects documented the NFRs:
  - 3 used templates (1 only for initial NFRs)
  - 1 used plain text (only for initial NFRs)

Opinion: NFRs are often described in non-measurable terms and with vague wordings [46]. Sabaliauskaite et al. reported that NFRs tend to be badly structured or vague [191]. Svensson et al. reported different situations in their case studies [41]. Olsson et al. reported that about half of NFRs considered in a case study were quantified [168]. This empirical study aligns with these observations, just 2 out of 13 respondents (**(B)** and **(D)**) providing some quantification level, which is even less than the mentioned above. **(B)** and **(D)** did it this way (*"You need to be able to provide arguments when discussing with the customer"* **(D)**). In fact, these two architects were also the only ones who maintained the requirement documentation up to date; **(J)** and **(K)** only documented the initial NFRs (*"At first, we wrote down some initial ideas for NFRs in natural language [...], but afterwards we did not keep track of any of them or of any other NFRs arising during the* *design process"* **(K)**). It seems natural to think that there is a relationship between measurability and continuous (or at least regular) update of their documentation but this link needs to be confirmed in further studies.

We observed that NFRs are more tacit or even hidden than documented, and when documented, their accuracy and timeliness is seriously compromised. This situation can be explained in terms of cost and benefit. One of them stated it in plain words: *"I rarely appropriately document my projects, basically because it costs money"* **(C)**. If practitioners do not perceive a clear benefit from shaping NFRs into a fully engineered artifact, as is the case of **(D)**, NFRs will remain elusive.

Answer: NFRs were not often documented, and even when documented, the documentation was not always precise and usually become desynchronized.

RQ4.4: How are NFRs validated?

The validation of system behavior against the elicited NFRs is a tough activity. Since every NFR is different in nature, the methods needed are distinct too. In spite of this fact, most of the architects of our study (11 out of 13) claimed that all NFRs had been satisfied by the end of the project, although, as mentioned by **(H)**, *"there is always room for improvement."* However, when asked how they had validated them, their answers were vague. The following comment by **(D)** is illustrative: *"compliance with some [not all] NFRs is only informally discussed with the client, since it is not easy to test."*

Eight interviewees performed some validation, but each one validated only one to three NFRs. Few types of NFRs were considered: performance efficiency (*"we ran load and stress tests to evaluate performance"* **(H)**); correctness (*"for each hour of coding we spent one hour testing for bugs"* **(A)**); usability (*"we made a prototype just to ensure client satisfaction with the interface"* **(K)**); and reliability (*"we have forced some errors to see what happens and control loss of data"* **(J)**). Notably, one highly relevant type of NFR, security, was not mentioned by any of the respondents. One respondent **(F)** was an extreme case of non-validation, noting: *"We wait for the client to complain. He will notice when something goes wrong"*. Although the response is of course unsatisfactory, it shows again (as in the case of documentation) how budget and time considerations may interfere with ideal engineering practices.

Summary of findings

- NFRs had been met by the end of the project:
  - 11 architects claimed that the NFRs had been met by the end of the project
  - 2 architects did not claim that all NFRs had been met by the end of the project
- Validated types of NFRs:
  - 1 architect validated three types of NFRs (reliability, efficiency, and accuracy)
  - 3 architects validated two types of NFRs (efficiency and accuracy; efficiency and usability; efficiency and reliability)
  - 4 architects validated one type of NFR (efficiency twice; accuracy; usability)
  - 1 architect did not validate any NFRs at all
  - 4 architects did not provide details on this point

Opinion: The 85% (11 out of 13) of interviewees that claimed satisfaction of all NFRs is a high percentage, much higher than the 60% reported in [41]. One could argue that this observation we got in our study contradicts the statement by Borg et al. saying that most NFR types are difficult to test due to their nature [46] but in fact it is not the case. On the contrary, it indicates the need to distinguish between the perception of NFR satisfactibility (85%) and the real validation (8 out of 13, i.e., 61%, and not for all types of NFRs).

Three of the four types of NFRs mentioned by interviewees as validated, belong to what Borg et al. name *system characteristic types*, which means NFRs directly related to the characteristics of the systems per se; they report that in their study, these system characteristics are considered properly tested most of the cases, whilst others like usability are often poorly tested [46].

The architect that behaved differently was interviewee **(D)**, who used formal techniques based on statistical analysis and simulation to check the system's reliability. Of course, this approach to validation is highly related to the domain of the project, an information system for aerospace, i.e. a critical domain. This observation aligns with a previous survey by Ali Babar et al. whose participants suggested that the approach to evaluation depends on the evaluation goals [4].

One of the findings of our study that may align with previous results is the link between documentation and validation. Borg et al. that said: *"when expressed in non-measurable terms testing is time-consuming or even impossible"* [46]. Since we had just 2 respondents expressing the NFRs in a measurable form, this may be one of the reasons behind the low level of validation performed.

Answer: NFRs were claimed to be mostly satisfied at the end of the project although just a few types were validated.

RQ5: What type of tool support for NFRs is used?

All the architects declared that no specific tools were used for NFR management. Taking the chance of the exploratory nature of semi-structured interviews, we asked the interviewees if they would be willing to accept some help in the form of a decision support tool to assist them in architectural decision-making. The main motivation was to explore the real expectations from practitioners for this kind of tools, and we took note for the development of our own method/tool, which are presented in Sections 7.2 and 7.3.

We found a very strong reaction (e.g., *"I do not believe in automatic things"* **(B)**, or *"I would not trust"* **(F)**) against an automated decision-making tool from 5 of the respondents. The others were not so reluctant but expressed several concerns. 4 of them expressed their opinion that such a decision-making tool is simply too difficult to build (*"it is hard for me to imagine that this can be done"* **(I)**). A way to fight against this effect mentioned by 2 of the respondents was that the tool suggested alternatives instead of making final decisions (*"the tool could show you possibilities that you have not envisaged"* **(C)**). Also some worried about the amount of information that the architect should provide to such a tool for getting informed decisions (*"all the time that I would need for thinking and introducing all the necessary information, would not pay"* **(F)**). If such a tool would exist, architects would require a clear justification of decisions (*"the critical point is the accuracy of the tool and the answer that it could give"* **(C)**).

Summary of findings

- Architects did not use any specific tool support for NFR management

Opinion: This was one of the most extreme results of the survey. Even tool support as reported in [43] about dependency management (one important issue when it comes to NFRs), was missing. For sure the answer to this research question uncovers an important challenge to be addressed jointly by researchers and practitioners.

Concerning tool support for decision-making, this issue was mentioned by Ali Babar et al. in relation to some industrial cases that use tool support for generation of design option by exploiting some architectural knowledge [7]. Our observation about the type of tool practitioners may adopt aligns with the position reported by Hoorn et al. [111]: architects do not fancy proactive or automated support; instead, we share the view by Borg et al. [46] that methods and tools supporting NFRs throughout the entire development process are needed.

Answer: software architects did not use any specific tool for NFR management.

RQ6.1: What types of decisions are driven by NFRs?

Analysis of the responses shows different categories of decisions driven by NFRs:

- *Architectural pattern.* Given the type of projects addressed, a layered architecture was the natural option, but it is worth mentioning that some interviewees explicitly justified the decision (*"We used a layered architecture to support later changes"* **(J)**).

- *Implementation strategies.* Several types of requirements may require strategies at a very detailed architectural level, either as a general design decision (*"We opted by single sign-on to improve integrability of different subsystems"* **(L1)**) or a detailed decision over some component (*"the tables of a data base were duplicated because the access time was too high"* **(A)**).

- *Cross decisions.* Some NFRs imply a decision that cuts across the full architecture (*"we prefer to use technologies we have already mastered"* **(L1)**. One recurrent matter is the use of third-party components and especially open-source software (OSS) (*"We wanted to have access to code for maintainability reasons, thus we opted by OSS solutions"* **(D)**).

- *Technological platforms.* NFRs may be satisfied by the selection of technologies. Similarly as before, they can be system-wide (*"We need high availability and this requirement was ensured only by Oracle"* **(K)**), or more localized (*"One of the queries was implemented directly in JDBC instead of Hibernate due to efficiency reasons"* **(H)**).

Opinion: We have observed that NFRs drive several types of architectural decisions. Here we can appreciate again the importance of non-technical NFRs, especially in cross decisions, and in the selection of particular implementing technologies. We did not find any empirical work that had an observation on this topic, but we could mention here that this observation is aligned with Chung et al. *"[NFRs...] play a critical role during system development, serving as selection criteria for choosing among myriads of alternative designs and ultimate implementations"* [63], and Zhu and Gorton *"The rationale behind each architecture decision is mostly about achieving certain NFRs"* [217].

Answer: all the studied types of architectural decisions had, in some case, the influence of a NFR.

RQ6.2: How is the architectural decision-making process?

When discussing the decision-making process, one particular aspect that emerged was the intertwining among technological decisions and the others. We found three different responses. Four of the architects (**(C)**, **(J)**, **(K)**, **(L2)**) said that non-technological decisions come first (*"The architect should* *manage which technologies are appropriate to cover the previously designed logical structure"* **(C)**). On the contrary, other four (**(A)**, **(H)**, **(G)**, **(L1)**) mentioned that fundamental technological decisions come first in place and the others should adapt to it (*"we had some limitations from the client, e.g., architecture based on OSS and Java"* **(H)**). The remaining five argued both types of decisions overlap and feed each other, which in fact could be considered a local twin-peak model [164] at the level of SA design.

Opinion: Again, what is done in practice does not match with the academia. In this study we can see that the top-down architecture design approach supported by the academia (e.g., first the selection of architectural patterns, then allocate the logical components, and finally the implementation with some technologies [100]) is only performed by four interviewees (30%). It is worth to mention that MDD, a development approach where this top-down development is very explicit, is not having the expected adoption by practitioners (see the first empirical study, Section 5.1). We think that we should not expect that architects change their way of doing the decision-making, any approach that comes from the academia should be flexible enough to handle this variety of decision-making processes. In this same direction there was a comment made by interviewee **(C)** *"computing is a tool that should adapt to the organization, not the other way around"*.

Answer: the architectural decision-making process is diverse.

RQ6.3: How NFRs influence architectural decisions?

Not all types of NFRs have the same influence in decision-making. We asked the respondents which were the NFR types that they took into account when making architectural decisions. We consolidated their answers using the ISO/IEC 25000 quality standard as unifying framework [116]. Important observations are:

- *Explicitness.* Some NFRs are considered even if not explicitly mentioned, e.g., *"I would never think of a system that it is not secure"* **(I)**. Often, these tacit NFRs have percolated into architects' mindset due to the features offered by the technologies used, e.g., *"We didn't thought about the security of the documents because it is done by the management system of SharePoint"* **(E)**.

- *Source.* Some requirements come directly from the development team or the architect (e.g., *"We were the ones that would maintain the system, so, it was important for us to ensure its maintainability"* **(B)**). These NFRs are closer to the decision-making process than those coming from the client, since the technical staff already thinks in terms of the solution compared to the problem-oriented style of clients.

- *Non-technicality.* Non-technical NFRs are those that do not refer directly to the intrinsic quality of software, but to the context of the system under analysis, e.g., license issues or cost [56]. We observed that they are considered essential by architects. The estimation from the responses is that about 40% of the NFRs considered by respondents in their projects were non-technical, that in some cases were considered of highest priority (*"Money rules and everything has to be adapted to it"* **(J)**).

- *Importance.* In RQ3, we analyzed architects' perception of NFRs importance. We cross-checked this information with the decision examples provided in the interviews and we observed that performance efficiency and maintainability were the types of NFRs that drove most of these examples, which partially coincide with the RQ3 results.

Opinion: There are some works in the literature that provide frameworks to document architectural decisions explicitly [206, 5, 203, 209], but, as we have seen in this study, in practice architects do not make explicit all architectural decisions. To improve this situation we have to, first, due to the pragmatism of architects we should show them the benefits of having explicit architectural decisions, and then we could provide them tools to facilitate the documenting task (some tools are already available, see Section 6.3).

As we have seen in RQ4.1, architects are the main source of NFRs. Since architects are doing both the requirement elicitation and the decision-making, it may happen that they start thinking in the solution instead of identifying the real need in first place, which is the kind of situation where they may skip a good alternative decision. A way to fight against this practice is the separation of the decision making from the requirements elicitation, for example having two different roles in the software company, the requirements engineer and the software architect, but as we have seen in RQ1 this solution seems hard to achieve because there are still many companies that do not have even a role for architects.

We have shown in RQ3 that non-technical NFRs are important for architects, and in consequence for the architectural decision-making. We should probably give more attention to this kind of requirements in future empirical studies and approaches for architectural decision-making.

As we mentioned, the importance of NFRs detected in RQ3 is somehow reflected in the architectural decisions provided as example during the interview. We ask them to provide an example of decision that was hard to achieve, so it may also happen that the important requirements for architects are also the ones that lead to complex decisions.

Answer: architectural decisions are influenced by several aspects of NFRs.

5.2.3 Discussion

As shown in this study, semi-structured interviews are a very useful tool for learning about current practices, not only because they explore the predetermined survey questions, but also because unexpected observations arise that, while not directly integrated in the study's main results, help to provide a more complete picture and trigger ideas for further studies. We may cite as clear example in our study the leading role of software architects as the main source of NFRs. For this particular issue, a reflection on the variability and importance of the software architect role depending on the type of project can be found at [50]. As a beneficial side-effect, after an hour or more of being interviewed, respondents learn about certain issues to which they are not usually exposed (e.g., NFR terminology) and some of them expressed their desire to collaborate again on future studies (as it has really happened). On the other hand, consolidation of results coming from qualitative studies is far more difficult than in the case of quantitative ones, but the knowledge gathered is very rich and a good input for both researchers and practitioners.

We tried to consolidate our findings with previous empirical studies. First, we observed that there are not so many empirical studies centered on NFRs. A review conducted by Svensson et al. [43] in 2010 reported on 18 empirical studies that have some relationship to our proposal. But in fact, none of the previous studies included the relationship among NFRs and architectural decisions in their research questions and thus available evidence is anecdotal, which makes our own study clearly differentiated. Whilst these studies were more focused in the NFR part, others explore SA issues more in depth, and especially van Heesch and Avgeriou [208] mentions at some moment relationships among NFRs and SAs, as already reported with respect to requirement elicitation (see RQ1 and RQ4.1).

Still we have made an effort to align our observations with those studies and we have found: some of our observations match with previous findings (e.g., software architects performed other duties in the projects; NFR elicitation is iterative), some others have not been reported before (e.g., software architects perceived NFRs as satisfied, independently of the light validation performed) and a few contradict previous observations (e.g., measurability of NFRs was poor). Also the analysis of the influence of the different types of NFRs was analyzed: we were able to find some coincidences (e.g., de la Vara et al. also referring to efficiency and usability as the most important [79]) but it was not easy to proceed with the necessary rigor as to compare results. On the one hand, the studies use different NFR classification schemes. On the other hand, the roles involved are often different therefore their consideration of NFR types differ too [43].

We concur with different authors (e.g., [43, 86]) about the need of conducting more empirical studies on the role of NFRs in software architecture. This is why we decided to perform a third empirical study related to this topic, in this case to a particular architectural style.

5.3 Third empirical study

One type of software system that has become popular in industry is that of service-based systems (SBSs). Service-orientation is a standard-based, technology-independent computing paradigm for distributed systems [167]. In this paper, we define service-based architecting as the architecting of systems that are assembled from individual software services, invoked through standardized communication models [137, 165]. SBSs can address multiple execution environments by separating the service description (interface) from the implementation of the service. Furthermore, SBSs facilitate the use of software services aligned with business drivers [69].

Quality attributes (QAs) are characteristics that affect the quality of software systems. This is because QAs are often not explicitly described by stakeholders. Furthermore, QAs exhibit trade-offs that need to be negotiated and resolved between stakeholders.

In SBSs QAs are difficult to achieve [34], because SBSs lack a central control of the system due to limited end-to-end visibility of services, unpredictable usage patterns of services and dynamic composition of systems [34]. Often, QAs cannot be achieved by tuning a system after the SBS is implemented. Instead, achieving QAs is a continuous activity that should be emphasized throughout the whole software development cycle [34].

In contrast to conventional software systems, the role of QAs in the context of SBSs has not yet been studied extensively. However, quality is a top challenge in SBSs engineering [104, 167]. Even though proposals for QAs in SBSs exist, there is a lack of empirical studies that investigate QAs in practice [149].

Using the GQM approach [35], the goal of our study is defined as follows: to analyze and characterize (purpose) the role of QAs (issue) in SBSs architecting (object) from the perspective of practitioners and researchers with practical architecting experience (viewpoint).

5.3.1 Design

We conducted a descriptive survey [133] to study how QAs are treated during SBSs architecting, rather than why QAs are treated in a certain way. We used purposive sampling [73] and required participants to have practical experience in architecting SBSs. A prerequisite to participate in the survey was to had the architect role in an SBS project.

Surveys collect qualitative and quantitative information to provide a *snapshot* of the current status related to a phenomenon [212]. To ensure rigor, repeatability and to reduce researcher bias, we designed a survey protocol following the template proposed for evidence-based software engineering[6]. Furthermore, the study itself followed the survey process proposed by Ciolkowski et al. [65] and included the steps of survey definition, design, implementation, execution, analysis and packaging. We refined these steps based on survey activities described by Pfleeger and Kitchenham [178].

Research questions

We defined three questions (see Table 5.6):

- RQ1, *How important are QAs compared to functionality when architecting SBSs?*, current literature, such as [36, 208, 207, 32], suggests that QAs, and NFRs, drive the design of software architectures. We are interested in finding out if this is also the case for SBSs, or if QAs are mainly treated as factors that suggest the use of a service-based solution in the first place but are not considered architecture drivers. This is because service-orientation claims to achieve *qualities*, such as interoperability, flexibility or reusability [84]. Answering this question helps practitioners understand if QAs require special attention when architecting SBSs, similarly to conventional systems, or if architecting SBSs allows architects to focus on functionality because Qa are somehow achieved by the SBSs (e.g., through its patterns and technologies).

- RQ2, *To what extent are QAs specific to application domains of SBSs?*, we aim at identifying information that can provide guidance for software architects through the architecting process by focusing on the QAs that are most important for a certain application domain (e.g., healthcare, telecommunication). This question helps decide what QAs to focus on when architecting of SBSs for particular domains.

- RQ3, *What kind of architectural decisions are used to address QAs in SBSs?*, investigates the transition from QAs to architectural decisions by relating QAs to the architecture decision types proposed by Kruchten [143]. Furthermore, we relate decisions to decision categories (ad-hoc decisions, pattern, and technology). Answers to RQ3 provide a first step towards the solution space, i.e., how to accommodate QAs when architecting SBSs.

[6]`http://www.dur.ac.uk/ebse/resources/templates/SurveyTemplate.pdf`

Table 5.6: Research questions of the third empirical study

ID	Research Question
RQ1	How important are QAs compared to functionality when architecting SBSs?
RQ2	To what extent are QAs specific to application domains of SBSs?
RQ3	What kind of architectural decisions are used to address QAs in SBSs?

Data preparation and collection

All survey questions[7] referred to one particular project that participants had worked on in the past. Some questions were optional and some mandatory. Structured questions could be answered using Likert-scale or pre-defined answer options [134], unstructured questions allowed numeric answers or free text. For some questions, more than one answer was possible and participants were asked to choose the best-fitting answer. For most questions, participants were allowed to provide comments to complement their answer. We included four types of questions:

- Questions about the profile of participants were used to ensure reliability and to support data analysis.
- Questions about the project helped assess if project specifics have an impact on the way QAs are treated.
- Questions about the QAs helped identify relevant QAs for different projects. For describing QAs we used a scenario-based approach to make QAs concrete [36] and to avoid misunderstandings when referring to QAs.
- Questions to analyze architectural decisions made to accommodate QAs were used to identify what decisions help achieve QAs during architecting. Questions to describe decisions were derived from the template to describe architectural decisions as proposed in [206].

Data were collected from May 2011 to September 2011 through an online questionnaire. It took about 20 minutes to complete.

[7]The survey questionnaire is available in the Appendix B.

Data analysis

To ensure the quality of the data obtained from the questionnaire, we applied sanity checks to find obvious errors in data. We used descriptive statistics to analyze the data [132]. Furthermore, we analyzed dependencies between the answers provided to different questions. Questions which resulted in free text were coded [156] and underwent content analysis [138].

Limitations of the study

We have analyzed the construct validity and the internal and external validity of this study.

Construct validity. This aspect of validity reflects to what extent the operational measures really represent what is investigated according to the research questions [213]. This study was supported by two main principles: rigorous planning of the study according to Oates [166], and establishment of protocols for data collection and data analysis. The protocol was reviewed by external reviewers. The survey was piloted both internally and using external participants from the target population.

Internal validity. There might have been confounding variables and other sources that could bias our results [65, 135]. To control variables, exclusion or randomization can be applied [208]. Exclusion means that participants who are not sufficiently experienced were excluded from the study. We ensured this by having a check question that only allowed participants with architecting responsibility in a project to proceed with the questionnaire. Randomization means that we used a sampling technique which leads to random participants. Furthermore, validity is subject to ambiguous and poorly phrased questions. To mitigate this risk, we piloted the data collection instrument in multiple iterations until potential respondents understood our questions and intentions. Another limitation is that participants might not have answered truthfully to the questions [208]. To address this problem, we made participation voluntary and anonymous. Furthermore, participants spent personal time on answering the questionnaire. We can therefore assume that those who volunteered to spend time have no reason to be dishonest [208]. Also, participants might not have had the same understanding as we required them (e.g., what is a decision, what is a QA); we tried to mitigate this through sanity checks as well as through piloting the questionnaire with members of the target population.

External validity. External validity is concerned with the problem of generalizing the results to the target population. We assume that our results are applicable to a population that meets the sampling criteria of our survey (i.e., architects with industrial experience in SBSs). Also, the fact that some participants had a background as researchers and not only as practitioners may have influenced the results as they may have a broader view on SBSs architecting. However, answers are not just influenced by the understanding of participants, but also the characteristics of companies and software projects in which participants worked. We provided a brief discussion of how company and project size affected the results. Furthermore, we only had a limited number of participants (31 responses). However, this is due to the fact that our survey targeted a very specific population and required participants with knowledge about QAs, experience with architecting SBSs, and involvement in a real project. The number of participants in our study (31 participants) is similar to the number of participants in other empirical studies on software architecture or software requirements (e.g., 11 software companies [42], 53 industrial software architects [208], 22 students [207], or 39 architects within one company [181]). Note that none of these studies were limited to SBSs only. A recent survey on non-functional properties in SOA included 29 participants [38].

5.3.2 Results

In this section we answer the research questions stated at the beginning of this paper and provide demographic data to contextualize the information obtained in this study.

Demographic Data

We obtained 31 valid responses.

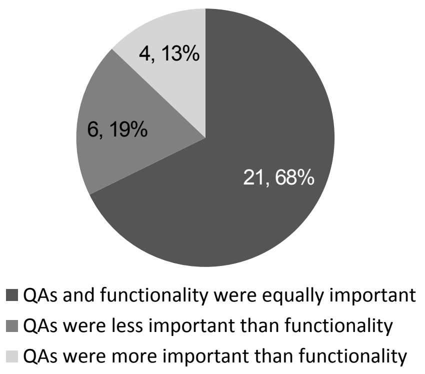
- We had participants from Europe (19, 61.3%), North America (7, 22.6%), Asia (2, 6.5%), Australia (2, 6.5%), and South America (1, 3.2%). The countries with most responses were Spain (7, 22.6%) and Germany (5, 16.1%).

- Eighteen participants (58%) had experience in both academia and industry. Ten participants (32%) had only experience in industry, whilst 3 (10%) were participants from academia who had worked on SBSs projects as part of their research).
- On average, participants with industrial experience had more than 12 years of such experience whilst participants with research experience had more than 5 years of experience in research related to SBSs.
- Participants played different roles in the selected projects, e.g., designer, consultant. In the results of RQ1 we relate roles to answers given by participants.
- The educational degrees of participants were uniformly distributed: PhD (7, 22.6%), MSc (9, 29.0%), BSc (8, 25.8%), and other (7, 22.6%). The majority of participants had degrees in computer science (24, 77.4%).
- From the total of participants with industrial experience (28 participants), 20 (71.4%) answered that their company has over 250 employees, 3 (10.7%) worked in companies with 50 - 250 employees, 4 participants (14.3%) in companies with 10 - 50 employees, and just one participant (3.6%) worked in a company with less than 10 employees.

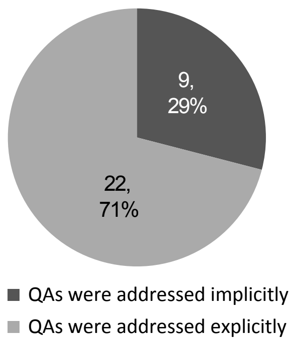
RQ1: How important are QAs compared to functionality when architecting SBSs?

Importance and explicitness of QAs. Figure 5.4a shows the response to the question about how important QAs were when architecting the system of the project selected by participants, compared to functionality. Functionality and QAs were considered equally important by most respondents. When asked whether QAs were considered implicitly or explicitly, most respondents (71%) answered: *explicitly* (see Figure 5.4b).

To study whether there is a dependency between the importance of QAs and their implicit or explicit nature, we created a cross-tabulation (Table 5.7). We observe that 18 respondents (58%) considered QAs and functionality equally important and at the same time made QAs explicit. On the other hand, in 6 cases (20%), QAs were not made explicit and QAs



(a) QAs compared to functionality



(b) Implicit/explicit nature of QAs

Figure 5.4: Classifications of QAs

Table 5.7: Importance of QAs and their implicit or explicit nature

	QAs explicit	QAs implicit	Total
QAs were as important as functionality	18 (58%)	3 (10%)	21 (68%)
QAs were less important than functionality	0 (0%)	6 (20%)	6 (19%)
QAs were more important than functionality	4 (12%)	0 (0%)	4 (13%)
Total	22 (71%)	9 (29%)	31 (100%)

were considered less important than functionality. In all 4 answers (12%) where QAs were more important than functionality, QAs were made explicit. There was no case in which QAs were made explicit and at the same time were considered less important than functionality. Fisher's exact test led to $p < 0.001$ which means that there is a statistically significant relationship between the importance of QAs and their implicit or explicit nature. Thus, there is a high probability that projects which treat functionality and QAs equally important also make QAs explicit. On the other hand, there is a very low probability that QAs would be treated equally important to functionality when QAs are considered implicitly.

Impact of training on how QAs are perceived. Table 5.8 and Table 5.9 show participant's training related to SBSs, and the importance of QAs, and their implicit or explicit nature.

Table 5.8: Importance of QAs and the training of participants

	Training	No training	Total
QAs were as important as functionality	15 (72%)	6 (60%)	21 (68%)
QAs were less important than functionality	3 (14%)	3 (30%)	6 (19%)
QAs were more important than functionality	3 (14%)	1 (10%)	4 (13%)
Total	21 (100%)	10 (100%)	31 (100%)

Table 5.9: Nature of QAs and the training received by participants

	Training	No training	Total
QAs explicit	16 (76%)	6 (60%)	22 (71%)
QAs implicit	5 (24%)	4 (40%)	9 (29%)
Total	21 (100%)	10 (100%)	31 (100%)

- *Training.* The majority of participants who received training treated QAs and functionality as equally important (15 responses, 72%). This corresponds to 48% of all participants. Three participants with training in SBSs considered QAs as less important than functionality (14%, or 10% over all participants) and 3 participants with training considered QAs more important than functionality (14%, or 10%). Also, 16 participants (76% or 52% over all participants) with training considered QAs explicitly, with just 5 participants (24%, or 16% over total) with training considering QAs implicitly.

- *No training.* Six participants with no training considered QAs and functionality as equally important (60%, or 20% of all participants); three participants without training (30%, or 10% of all participants) considered QAs as less important compared to functionality, and 1 participant without training (10%, or 3.6% of all participants) considered QAs as more important than functionality. Furthermore, 6 participants with no training (60%, or 20% of the total number of participants) considered QAs explicitly, whereas 4 (40%, or 13% of all participants) considered them implicitly.

However, Fisher's exact test did not reveal any significant dependency between training and the importance of QAs ($p = 0.622$). Similarly, there is

Table 5.10: Importance of QAs and the role of participants

	Architect	Other	Total
QAs were as important as functionality	14 (82%)	6 (55%)	20 (71%)
QAs were less important than functionality	2 (12%)	3 (27%)	5 (18%)
QAs were more important than functionality	1 (6%)	2 (18%)	3 (11%)
Total	17 (100%)	11 (100%)	28 (100%)

Table 5.11: Nature of QAs and the role of participants

	Architect	Other	Total
QAs explicit	12 (71%)	8 (73%)	20 (71%)
QAs implicit	5 (29%)	3 (27%)	8 (29%)
Total	17 (100%)	11 (100%)	28 (100%)

no statistically significant dependency between training and the implicit or explicit nature of QAs ($p = 0.417$). This means, there might be dependencies but given the p-value we cannot make any statistically significant claim.

Impact of role on how QAs are perceived. Even though all participants had architecting responsibilities in the project for which they answered the questions, they had different roles. Architects were the majority (17 participants or 55% of all participants). Additional roles included 3 project managers (10%), 2 developers (7%), and 1 participant of each of the following roles: consultant, quality engineer, analyst, industrial researcher, unit manager and standards developer. Cross-tabulations are shown in Table 5.10 and Table 5.11. Three participants did not provide any role. Thus, the total number in Table 5.10 and Table 5.11 is 28 instead of 31. Fisher's exact test indicated a dependency between the role of participants and how they judged the importance of QAs ($p = 0.078$). Given the number of architects that considered QAs as equally important compared to functionality, this dependency means that architects and designers tend to treat QAs and functionality equally important. Furthermore, most architects and designers treated QAs explicitly (71% of all architects). However, Fisher's exact test ($p = 0.151$) did not reveal any significant dependency between the role of participants and how they treated QAs.

Additional observations

- We did not find any indicator that participants with more years of industrial experience treat QAs as more (or less) important than functionality, or consider QAs implicitly or explicitly. Also, there was no difference between general software engineering work experience and work experience related to SBSs architecting when considering the impact of experience on the importance or implicit / explicit nature of QAs.

- Project size in person-months does not have any significant relationship to the importance of QAs, nor to the implicit or explicit nature of QAs. On the other hand, the company size has an impact on how QAs are perceived (Fisher's exact test led to $p = 0.115$) and treated ($p = 0.022$). The larger the company, the higher the probability that QAs are treated explicitly rather than implicitly. Similarly, the larger the company, the higher the probability that QAs and functionality were treated equally important.

- We have not found any relationship between the type of the project selected by the participants (single service, a complete SBSs system, or systems using services as part of conventional systems, i.e., hybrid systems) and the importance and implicit / explicit nature of QAs.

- When studying the relationship between the importance of QAs and their implicit / explicit nature, and the reason why SOA was chosen as design paradigm, we found that there is no statistically significant relationship. However, we found that 23% of participants treated QAs explicitly and at the same time indicated that certain QA suggested the use of service-orientation. Similarly, 26% of the participants treated QAs and functionality equally important and indicated that certain QA suggested the use of a service-based solution. This means, when QAs suggested the use of SOA, QAs tend to be treated explicitly and equally important as functionality. However, even though QAs were the driver for choosing service-orientation as design paradigm, during the architecting phase QAs were not considered more important than functionality.

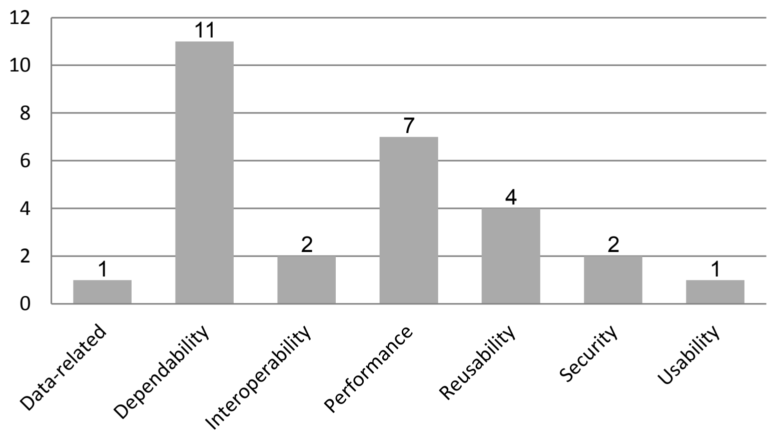


Figure 5.5: Frequency distribution of QA

Answer: the majority of software architects perceived QAs at the same level as functional requirements, and also the majority of architects make the QAs explicit in their projects. We found no relation of this fact to the training, role, or experience of the participant, also no relation was found with the project size and type.

RQ2: To what extent are QAs specific to application domains of SBSs?

To answer RQ2, we used responses to the question about the most important QAs that participants had experienced. During analysis we mapped all QAs stated by participants in terms of scenarios to QAs for SBSs as defined by the S-Cube quality model [29]. This was done through content analysis where combinations of three researchers categorized each QA. Figure 5.5 shows the frequency distribution of QAs. We grouped data-related quality attributes from the S-Cube quality model (data reliability, completeness, accuracy, integrity, validity). Dependability and performance are the most frequently addressed QAs. Note that not all participants provided a complete scenario. Thus, the total number of types in Figure 5.5 and in Table 5.12 is 28.

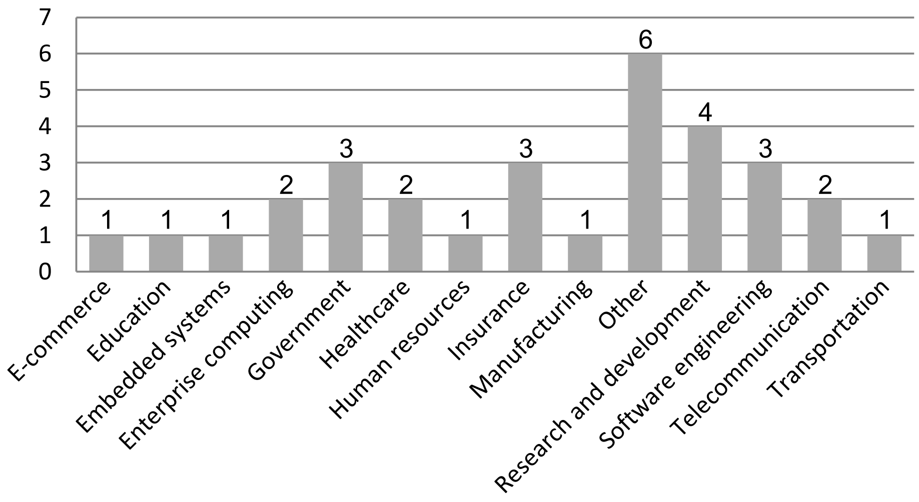


Figure 5.6: Frequency distribution of application domains

Figure 5.6 shows the frequency distribution of domains of projects that participants had worked in. The category *Other* includes domains such as aerospace, real estate and social networking. Fisher's exact test did not reveal any relationship between QAs and domains ($p = 0.456$). Also, cross-tabbing domains and QAs did not show a QA that would be addressed more than twice in a domain. This means, we could not identify any QA that would be particularly relevant for a certain domain. In Table 5.12 we show the cross-tabulation between the QAs and their importance (see RQ1). Except for dependability and performance which tend to be considered more important than functionality, there is no trend for a relationship between other QAs and their importance ($p = 0.983$). Also, there is no relationship between the QAs and if QAs has been treated explicitly or implicitly ($p = 0.837$). The only trend that we observed is that dependability and performance tend to be treated explicitly.

Answer: with the gathered data we have not found any relation between QAs and application domains, but we have found that dependability and performance are important QAs for SBSs.

Table 5.12: QAs and their importance

	Equal[1]	Less[2]	More[3]	Total
Data-related	1 (5.3%)	0 (0%)	0 (0%)	1 (3.6%)
Dependability	7 (36.8%)	2 (33.3%)	2 (66.7%)	11 (39.3%)
Interoperability	1 (5.3%)	1 (16.7%)	0 (0%)	2 (7.1%)
Performance	5 (26.3%)	1 (16.7%)	1 (33.3%)	7 (25.0%)
Reusability	3 (15.8%)	1 (16.7%)	0 (0%)	4 (14.3%)
Security	1 (5.3%)	1 (16.7%)	0 (0%)	2 (7.1%)
Usability	1 (5.3%)	0 (0%)	0 (0%)	1 (3.6%)
Total	19 (100%)	6 (100%)	3 (100%)	28 (100%)

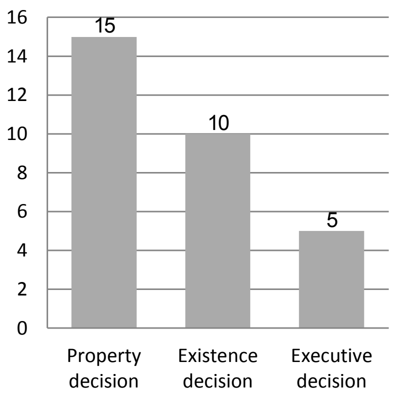
[1] QAs were as important as functionality
[2] QAs were less important than functionality
[3] QAs were more important than functionality

RQ3: What kind of architectural decisions are used to address QAs in SBSs?

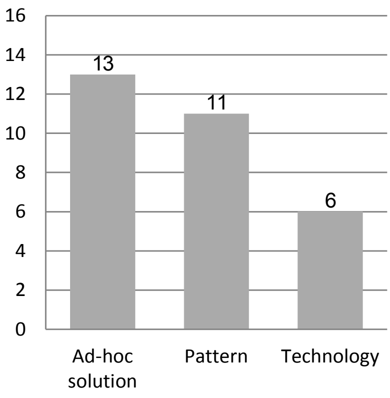
We used two classifications to differentiate the kinds of decisions. First, we used Kruchten's taxonomy of decision types [143]. And second, we classified decisions based on the following decision categories:

- *Ad-hoc:* Solution that is specific to a concrete problem of the project (e.g., the architect decides to create a separate service to store sensitive information about the users to improve the security of the system).
- *Pattern:* Reusable and widely-known architectural solution (e.g., the decision to use of the Model-View-Controller pattern for structuring the user interaction).
- *Technology:* A piece of implemented software that fulfills some required functionality (e.g., the use PostgreSQL instead of other DBMS).

Assigning decisions made to accommodate QAs to types and categories of decisions was made based on a content analysis involving all authors. As Figure 5.7a shows, 15 decisions (50.0%) where classified as property decisions, 10 decisions (33.3%) as existence decisions, and 5 decisions (16.7%) as



(a) Decision types



(b) Decisions categories

Figure 5.7: Classifications of decisions

executive decisions. Furthermore, as Figure 5.7b shows, 13 decisions (43.3%) were classified as ad-hoc, 11 decisions (36.7%) as pattern, and 6 decisions (19.4%) as technology. Only 1 decision was not classified because the participant did not provide a description for it. We investigated relationships and found a correlation between decision types and decision categories (Fisher's exact test: $p = 0.018$): 83.3% of technology decisions are existence decisions, 69.2% of the ad-hoc decisions are property decisions, and 54.5% of pattern decisions are also property decisions.

QAs and decision classification. We tried to find correlations between the decision classifications and the QA mentioned by the participants. The results are not significant. We obtained a Fisher's exact test of $p = 0.835$ and $p = 0.741$ for decision types and decision categories, respectively.

QAs treatment and decision documentation. As part of analyzing the types of decisions made to accommodate QAs, we also studied if these decisions were actually documented or treated implicitly. This is important as not documenting decisions can lead to problems later during SBSs architecting. There is a correlation between treating QAs explicitly as requirements and documenting decisions (Fisher's exact test: $p = 0.022$). All participants that treated QAs explicitly also documented the decisions made in order to accommodate this QA. Also, all participants that did not document decisions treated QAs implicitly. The complete relation between the

Table 5.13: Nature of QAs and documentation

	Not documented	Documented	Total
QAs explicit	0 (0%)	18 (78.3%)	18 (69.2%)
QAs implicit	3 (100%)	5 (21.7%)	8 (30.8%)
Total	3 (100%)	23 (100%)	26 (100%)

Table 5.14: Importance QAs and documentation

	Not doc.	Documented	Total
QAs were as important as functionality	1 (33.3%)	17 (73.9%)	18 (69.2%)
QAs were less important than functionality	2 (66.7%)	3 (13.0%)	5 (19.2%)
QAs were more important than functionality	0 (0%)	3 (13.0%)	3 (11.5%)
Total	3 (100%)	23 (100%)	26 (100%)

implicit and explicit nature of QAs and their documentation is shown in Table 5.13. Furthermore, we found a relationship between the importance of a QA and if decisions related to accommodating this QA have been documented ($p = 0.112$). The complete relationships are shown in Table 5.14. Note that the total number of responses in Table 5.13 and Table 5.14 is 26. This is because not all participants provided information about the degree of documentation of their architecture decisions.

Answer: all types and categories of architectural decisions are present in SBSs. Remarkably the property type of decisions which normally are related to QAs have predominance, this could be an indicator that QAs are specially relevant in SBSs. Ad-hoc solution is the predominant category, which may indicate that the SBSs are is still not mature enough, since the patterns and technologies do not solve most of the architectural decisions.

5.3.3 Discussion

Overall, literature argues that QAs are important and are a major challenge when architecting SBSs [104]. Our study showed that 71% of the participants indicated that QAs were treated explicitly. This could be an indicator that special attention is paid to QAs because they pose a major challenge. On the other hand, general literature about software architecting and design claim that QAs drive the architecture [36, 173]. Our study cannot confirm or reject this claim. We found that QAs were rarely more important than functionality. However, stating that QAs drive the architecture is different from stating that QAs are more important than functionality. In fact, both (QAs and functionality) are important but architecting methods (e.g., [32]) usually start with analyzing QAs as key drivers which then help select architecture solutions, such as patterns and tactics. Also, our study showed that in 29% of all projects, certain QA suggested the use of a service-based solution. This could be an indicator that QAs were treated as global influence factors or architectural drivers for high-level architectural decisions. It also indicates that using a service-based solution is not only a technology-driven decision but has sound rationale based on QAs.

We found that the majority of participants treated QAs and functionality as equally important in the context of SBSs. A recent study by van Heesch and Avgeriou on the reasoning process of professional software architects revealed that most architects consider functionality as important or very important, but QAs as clearly more important than functional requirements [208]. However, van Heesch and Avgeriou also acknowledge that one of the most important things in architectural decision-making is to treat both functional requirements and QAs as first-class concerns.

In another study, van Heesch and Avgeriou studied the reasoning process of *naïve* architects (e.g., novice or junior architects) [207]. In their study, more than 80% of participants indicated that NFRs play a prominent role during architecting. A similar result can be found in our study with practitioners in the context of SBSs as only 19% of our participants indicated that QAs were less important than functionality. Our results show that only 13% of participants treated QAs as more important than functionality (i.e., QAs did not play the most prominent role for 13% of the participants).

In [38] the authors conducted a survey to evaluate a catalog of non-functional properties for SOA, from the perspective of service consumers. The design and goal of this survey differed to ours. For example, the survey required participants to evaluate a catalog with 19 non-functional properties. These QAs were prescribed, rather than elicited from participants as in our study. As a result, the study refined definitions of non-functional properties of their original catalog and renamed some properties. Furthermore, some properties were added to or removed from the original catalog. The study found that security was prioritized as being absolutely essential in a quality model for SOA. However, our study showed that security only occurred in two projects. Also, from the seven QAs (Figure 5.5) found in our study, only three (performance, security, usability) are also included in the list of non-functional properties for SOA proposed as a result of the survey presented in [38]. Interoperability, a QA found in our study was considered as relevant for service providers, not for service consumers in [38]. Interestingly, reusability or dependability, two main features of SBSs were not found to be relevant non-functional characteristic in SOA in [38].

A study in the embedded systems industry [41] studied how NFRs are handled in practice. The study involved interviews with five product managers and five project leaders from five companies. Even though the research questions and the domain of this study were different than our research questions, the study found that usability and performance are the most important quality aspects. In contrast, our study found dependability and performance as the most important QA, with usability being the least important QA. The difference in the importance of usability could be due to the nature of embedded systems versus SBSs: in embedded systems user interfaces receive great attention and can determine the acceptance of a system by end users; in SBSs the composition of a system by third-party services imposes considerable challenges on the system dependability.

Poort et al. [181] studied NFRs as seen by architects. The study found that as long as architects are aware of NFRs, they do not adversely affect project success. This is in line with our results that most participants consider QAs explicitly and at least equally important as functionality. Furthermore, Poort et al. found that modifiability requires special attention if it is business critical. In contrast, we did not find any indicator that modifiability could threaten project success. This may be due to the fact that SBSs are considered highly malleable and reconfigurable systems by definition.

5.4 Conclusions

In this chapter we have presented the results of three empirical studies. The first one has been an instrument to show the software development practices, and in particular, we showed the impact of NFRs in the software development practices. In the second empirical study we have presented the results about how software architects deal with NFRs in practice. We have focused our research questions on three activities: elicitation, documentation and validation; and on three other issues: terminology, ranking of types and tool support. In the third empirical study we presented the results of a survey to study the role of QAs in the context of SBSs architecting.

In Table 5.15 we can see a selection of relevant observations made in the three presented empirical studies in this chapter. We can see that, for instance, the importance of NFRs is reported in the three studies, in particular when referring to software architecture, but looking at the results it seems that this importance is not tied to the particular domain or the particular architectural style used. Both in the first and in the second studies, the respondents declared that they are not interested in automatic tools to handle NFRs, but instead would want to have tools supporting the tasks related to NFRs. In the second study, we observed that non-technical and technical NFRs are considered of equal relevance, and in the third study, the architects said that functional and NFRs are of equal relevance. From both observations, we can conclude that, all kinds of requirements are considered relevant by the participating roles and studied domains, i.e., architects will take into account all kinds of requirements in architectural decision making.

One of the most recurrent question in the three studies, was to rate the most important types of NFRs. Table 5.16 shows the different results obtained in the three studies. Disregarding the fact that the studies do not target the same population, we can see that there are some types of NFRs that appear in the three studies: reusability/maintainability and efficiency/performance; while there are two types of NFRs that appear in the first two empirical studies but not in the third: reliability and usability. There is much variability in these results: respondents were thinking in specific projects, and they were subjective in their answers (e.g., a respondent may be very concerned about security and s/he will find means to justify that security is the most important NFR in most of the projects). This variability means that we cannot establish a strict order of importance, we can only indicate a general tendency.

Table 5.15: Relevant observations from the three empirical studies

Study	Observation
First	NFRs are important for practitioners, almost all the types of NFRs were considered important by the respondents
First	The importance of NFRs is not hardly conditioned by the use of a particular architectural style
First	Practitioners want to have the last word in the architectural decisions made. Automatic tool will not suit their needs
First	MDD techniques are far from being adopted by practitioners
Second	NFRs were mostly elicited by architects
Second	Software architects considered non-technical NFRs as relevant as technical NFRs
Second	Software architects performed other duties in the projects, and software architects did not exist as a differentiated role
Second	NFR elicitation is iterative, NFRs were not often documented, and just a few types of NFRs were validated
Second	Software architects did not want automatic NFR-based decision-making tools but accepted architect-driven tools
Second	We had indicators that the methods and techniques coming from the research community have not been adopted by practitioners
Third	QAs are often considered as important as functionality. QAs are often treated explicitly
Third	We could not identify QAs that would occur in particular domains
Third	Even though many QAs are treated explicitly, during architecture design they are often addressed through ad-hoc decisions
Third	We did not find a correlation between QAs and the type or category of decision that was made in order to accommodate a QA

Table 5.16: Most important types of NFRs from the three studies

Study	Top type of NFR	Second order	Third order
First	Reliability	Maintainability Efficiency	Usability Reusability
Second	Usability	Performance Reliability Maintainability	Security
Third	Dependability	Performance	Reusability

Part III

Arteon, Quark, and ArchiTech

Chapter 6

State of the art

This state of the art is based on the state of the art presented in the PhD thesis proposal [10], and has been updated with relevant works published after the proposal.

In the last decade, the essential role that Architectural Knowledge (AK) [76] plays in any software development process has been recognized. More concretely, the concept of Architectural Decisions (ADs, also known as Architectural Design Decisions, ADDs) has been identified as a key concept of AK (see quote bellow). The importance of AD concept is mentioned in many publications (e.g., [121, 143, 206]). This advance in software architecture has triggered several actions in the research community such as having dedicated tracks in software architecture conferences, special issues in journals, and dedicated events (e.g., the workshop on Sharing and Reusing Architectural Knowledge).

"[ADs...] are defined as those Decisions that are assumed to influence the Architectural Design and can be enforced upon this Architectural Design, possibly leading to new Concerns that result in a need for taking subsequent decisions. [...] Note that architectural design decisions need not necessarily be "invented" by the architect himself; architectural patterns, styles, and tactics are examples of architectural design decisions

(or, more precisely, alternatives) that are readily available from other sources", R. de Boer et al., "Architectural Knowledge: Getting to the Core", 2007 [76].

Related to AK, there are several topics that are of especial relevance for this thesis:

- *Architectural Knowledge ontologies.* Ontologies are the principal way of representing knowledge, and AK is no exception. These ontologies are presented is some cases, as models or metamodels, but always as a way to organize the elements and concepts that are relevant to AK. In this thesis is presented an ontology for AK, Arteon (see Section 7.1), which was inspired in some of the works presented in this part of the state of the art.
- *Software Architectural Design Methods.* The design of the tool produced in this thesis, ArchiTech, was supported by a Software Architectural Design Method (SADM), Quark (see Section 7.2). SADMs are methods that help the architect to derive the software architecture from the software requirements [86]. Before defining Quark, we studied the available SADMs in the field (e.g., Attribute-Driven Design Method [32]).
- *Architectural Knowledge tools.* As result of this thesis one tools was produced to manage AK and assist software architects in architectural decision making, ArchiTech (see Section 7.3). In this part, the tools found to manage AK are analyzed.

These topics are studied in the following sections.

6.1 Architectural Knowledge ontologies

Several ontologies have been published to represent AK, each with different nuances, but most of them making a special emphasis on the notion of ADs. One particularly relevant work in this direction is the ontology proposed by P. Kruchten et al. [145] to describe the types of ADs (it was previously published in 2004 [143]). In this taxonomy ADs are classified into: existence decisions, property decisions, and the executive decisions:

Property decision: "A property decision states an enduring, overarching trait or quality of the system. Property decisions can be design rules or guidelines (when expressed positively) or design constraints (when expressed negatively), as some trait that the system will not exhibit", e.g., "all domain-related classes are defined in the Layer."

Existence decision: "An existence decision states that some element / artifact will positively show up, i.e., will exist in the systems' design or implementation", e.g., "the logical view is organized in 3 layers."

Executive decision: "These are the decisions that do not relate directly to the design elements or their qualities, but are driven more by the business environment (financial), and affect the development process (methodological), the people (education and training), the organization, and to a large extend the choices of technologies and tools", e.g., "system is developed using J2EE."

The following are works where a conceptualization of AK was presented. Being flexible, we may understand these conceptualizations as ontologies even that they are named as metamodels or other terms by their authors[1]:

- A. Jansen et al. [121] presented a metamodel that put ADs as the central concept. The metamodel is divided in three parts: composition model, architectural model, and design decision model. ADs are described by means of design fragments. Other relevant concepts that appear in this metamodel are: connector, interface and port.
- R. de Boer et al. [76] presented a model to represent the "core" of AK. This work defines ADs as alternatives. Other relevant concepts that appear in this model are: stakeholders, activities, and artifacts. The model is represented using a custom made modeling language.
- P. Avgeriou et al. [29] presented a conceptual model to represent an AD. This work defines decisions as options. In this work decisions are

[1]Sometimes the terms used to refer to these conceptualizations (model, metamodel, ontology, taxonomy, etc.) are used by convenience (e.g., the target audience) instead of their actual meaning. The differences between these terms are described in: `http://infogrid.org/trac/wiki/Reference/PidcockArticle`

related to rationale, issues, and concerns. This model pretend to be an extension to the ISO/IEC/(IEEE) 42010 [118].

- R. Capilla et al. [54] presented a metamodel for architecting, managing and evolving architectural design decisions. This work divides the concepts in three groups: project model, architecture, and decision model. The project model includes concepts such as stakeholders, iterations, requirements, and views of the architecture. The part named as architecture have concepts such as variation points, patterns and styles. The decision model includes concepts such as constraints, dependencies, decision-making activity, and assumptions rationale.

- C. López et al. [147] presented an ontology that describes Soft-goal Interdependencies Graphs (SIGs) semantics concepts to represent NFR and design rationale knowledge. This ontology does not include architectural concepts, but the concepts related to interdependency, argumentation, and decomposition. The ontology is described using the OWL language.

- A. Akerman and J. Tyree [1] presented an ontology that focus on ADs. The ontology is divided in four parts: architecture assets, architecture decisions, stakeholder concerns, and architecture roadmap. The architecture assets concepts offer an accurate description of the structure of the architecture. Concerns are addressed by ADs, these, in turn, are implemented in roadmaps. The ontology is represented in UML.

- ArchVoc [31] is an ontology for representing the architectural vocabulary. The terminology is classified in three main categories: architectural description (e.g., frameworks, views, and viewpoints), architectural design (patterns, styles, methodologies, etc.), and architectural requirements (non-functional requirements, and scenarios).

- Pahl et al. [174] presented an ontology that focused on components and connectors as a general way to describe architectural styles. This ontology uses a precise notation because the final objective is to provide a modeling language for architectural styles.

- Dermeval et al. [80] presented a metamodel that relates ADs, $i*$ and ACME [94]. In their metamodel we can observe that decisions are

Table 6.1: Comparison of AK conceptualizations

Ref.	ADs are...	ADs are related with...	Modular[1]	Other aspects covered
[121]	Design fragments	Composition techniques	Yes	Architecture structure
[76]	Alternatives	Concerns	No	Design process
[29]	Options	Rationale, issues, concerns	No	None
[54]	Patterns, styles	Constraints, dependencies	Yes	Project, architecture
[147]	N/A	N/A	Yes	NFRs
[1]	Alternatives	Concerns, assumptions	Yes	Architecture structure
[31]	N/A	N/A	Yes	Architecture structure
[174]	N/A	N/A	No	Architectural styles
[80]	Alternatives	Rationale, consequences	Yes	ACME, *i**
Arteon[2]	*Decisions*	*Quality attributes*	*Yes*	*Architecture structure*

[1] The conceptualization is described in different modules or there is some kind of separation.
[2] Our ontology, Arteon, is explained in Section 7.1.

alternatives, and the special relevance of NFRs in their approach. As outcome of a decision they produce a design fragment, which is represented using ACME.

6.1.1 Analysis

In Table 6.1 there is a summary of the works to represent AK mentioned in this section. The concept of alternative appear in three of the studied works [76, 29, 1], in this same three works also appear the concern concept related to ADs. These coincidences may be consequence of collaborations between the authors, it is also worth to mention that they are very near in time. Most of the works present the concepts separated in different aspects, this is also a recommended practice when designing ontologies. Five works considered relevant to include concepts related to the structure of the architecture as part of the AK (in the case of [29] the intention is not to represent AK, only the part related to ADs).

None of the studied conceptualizations fulfills the underlying needs of a computer-aided support system to make architectural decisions: a computer oriented formalism and enough detail to design an architecture. In this thesis we try to fulfill these needs, this is the reason why we designed an ontology, Arteon (see Chapter 7). Arteon is inspired in many of the studied ontologies and complemented with the required detail and formalism to be used in a computer-aided support system context.

6.2 Software architectural design methods

In this section are analyzed some of the Software Architecture Design Methods (SADMs) available in the literature, and more concretely is important for the contents of this thesis how these methods deal with NFRs.

One of the principal producers of this type of methods is the Software Engineering Institute (SEI). SEI has created several design and analysis methods: SAAM [126], ATAM [128], CBAM, QAWs, QUASAR, ADD [32], ARID. Documentation for all of them can be found in SEI website[2]. The most relevant ones, in relation to the contents of this thesis, are ADD and ATAM.

- Architecture Tradeoff Analysis Method (ATAM, R. Kazman et al.) [128] is a methodology that evolved from Software Architecture Analysis Method (SAAM, 1996). It is a method to understand the tradeoffs of the architectures of software-intensive systems. This method analyzes the architecture for several quality attributes (e.g., security, performance, etc.). The method guides the design decisions that have an impact on quality attributes. It is a spiral method, consisting in four phases: requirements elicitation including constraints, architectural views, modeling and analysis, and identification of tradeoffs.

- Attribute-Driven Design Method (ADD, F. Bachmann and L. Bass) [32]. A second version of this method was published in 2007 (available in the SEI website). ADD is a method to design the software architecture of a system based on quality goals for the system. The method is extensible for any quality attributes but has been particularly elaborated for the attributes of performance, modifiability, security, reliability, availability and usability. The method considers three architectural views: module view, component and connector view, and deployment view. The method consist in decomposing the system recursively into subsystems and then into components.

We did several searches in academic databases (e.g., Google Scholar, ISI Web of Science, etc.), complemented with other methods that we were already aware of to build a list of SADMs:

[2]www.sei.cmu.edu/architecture

- Quality Atribute-oriented Software ARchitecture (QASAR, J. Bosch) [47] is a method performed in three steps. First, the functional requirements are implemented in components, then the architecture is analyzed to decide whether the NFRs are fulfilled or not, and in the third step the architecture is adapted to be in conformance with the NFRs. The second and third steps are repeated till the whole system is in conformance.

- Quality-driven Architecture Design and Analysis (QADA, M. Matinlassi et al.) [151] is a set of methods that include a method for selecting an appropriate family architecture approach, a method for quality-driven architecture design, a method for evaluating the maturity and quality of the family architecture, and a technique for representing variation points in the family architecture.

- Quality Achievement at the Architectural Level (AQUA, H. Choi et al.) [60] is a method that provides software architects means for achieving quality attributes at the architectural level. AQUA involves two kinds of activities, which are architectural evaluation and transformation.

- A. Bertolino et al. [44] presented an approach to automate the architecture design and implementation. The method starts from requirements in Natural Language (NL). The authors say that they want to integrate several existing tools to accomplish the task: QuARS (Quality Analyzer for Requirements Specifications, tool to obtain requirements from NL specifications), ModTest (a model-checking tool), and Cow Suite (a testing tool).

- T. Al-Naeem et al. [2] proposed a method centered on the decision making process, but not on generating the architecture. The method uses Analytic Hierarchy Process (AHP) to score each alternative decision. The author says that other Multiple Attribute Decision Making (MADM) methods could be used instead of AHP.

- Tang et al. proposed the AREL method [202] to improve the traceability of ADs by linking them to the design rationale. They also propose a conceptual model to manage this rationale, and the concern for software quality during the architecture design, but they do not describe which is the reasoning method used (if any).

- Montero and Navarro proposed ATRIUM method [159], Architecture Traced from RequIrements applying a Unified Methodology. ATRIUM is a methodology based in the MDD approach, its intention is to guide the architects in the definition of the architecture, the method considers both functional and non-functional requirements.

- L. Chung et al. [61] proposed a framework, Proteus, to develop software architectures considering NFRs in goal-oriented notation, using NFR Framework [63].

- Tang et al. proposed the AREL method [202] to improve the traceability of ADs by linking them to the design rationale.

Many other SADMs are improvements or specializations of the previous:

- S. Bode et al. [45] presented a method based on QASAR to design the system's security architecture. The authors state that they considered methods form software engineering and security engineering to deal with security requirements.

- S. Kim et al. [129] presented a method that is based on architectural tactics. Architectural tactics are explained in L. Bass et al. book "Software architecture in practice, second edition" [36], they are basically reusable pieces of the architecture. This method uses feature models to generate the architecture automatically. It is very similar to a product line for architectures.

- D. Perovich et al. [177] presented a method to design software architectures, ATRIUM, using MDD considering quality aspects (based on ADD method). In this case they use a "megamodel" (a model composed of models) to represent the software architecture. The method uses feature models to construct the architecture.

- E. Niemelä and A. Immonen [161] presented the QRF method, this method extends QADA by providing a systematic method for eliciting and defining NFRs, tracing and mapping these requirements to architectural models and for enabling quality evaluation.

Table 6.2: Comparison of SADMs

Ref.	Name	Kind of NFR	Computer-aided	Based on...
[128]	ATAM	Quality aspects	No	SAAM
[32]	ADD	Quality aspects	No	None
[47]	QASAR	Quality aspects	No	None
[151]	QADA	Quality aspects	Yes, as product lines	None
[60]	AQUA	Quality aspects	Somehow, transformations	None
[44]	No name	Requirements	Yes, no details	None
[2]	ArchDesigner	Quality aspects	Yes, limited to decisions	None
[202]	AREL	Concerns	No	None
[159]	ATRIUM	Any NFR	Yes, MDD method	None
[61]	Proteus	Any NFRs	No	NFR Framework
[45]	No name	Security	No	QASAR
[129]	No name	Any NFR	Yes, as product lines	ArchDesigner
[177]	No name	Any NFR	Yes, MDD method	ADD
[161]	QRF	Quality aspects	Yes, as product lines	QADA
[33]	No name	Quality aspects	Somehow, decision making	ADDv1
	Quark[1]	*Quality aspects*	*Yes*	*Empirical evidence*

[1] Our SADM, Quark, is explained in Section 7.2.

- F. Bachmann et al. [33] proposed an improved reasoning framework for ADD method (first version). The authors distinguish between architectural model and quality attribute model and characterize the actions that a reasoning framework undertakes as basic architectural transformations.

Finally, it is worth mentioning one interesting work was published by Hofmeister et al. in 2007 [110]. Their intention was to produce a general model of architectural design methods based on empirical observation. Their main result is a model with three activities:

- Architectural analysis, *"serves to define the problems the architecture must solve"*.

- Architectural synthesis, *"proposes architecture solutions to a set of architectural significant requirements"*.

- Architectural evaluation, *"ensures that the architectural design decisions made are the right ones"*.

6.2.1 Analysis

The Table 6.2 summarizes the studied SADMs. There are many SADMs that use the ideas behind product lines to design architectures. It is interesting to see that almost all are capable to deal with quality aspects or NFRs in general, not limiting to a particular type as it happens in some MDD approaches. It seems to be more common to speak about quality aspects than NFRs in this area. SADMs that are based on other SADMs are more specific and are oriented to facilitate the automation of the method. There are many SADMs that consider NFRs and some of them are able to generate an architecture in a semi-automatic way.

6.3 Architectural Knowledge tools

There are, already, many tools to manage AK. This may be the reason why, as far as we now, there are three papers published to compare tools related with AK:

- A. Tang et al. [202]: in this work is published a comparative of five AK tools, with especial emphasis in the name used for architectural concepts.
- K. Henttonen and M. Matinlassi [109]: this work is focused on Open Source Software (OSS) based tools for AK.
- M. Shahin et al [194]: this work compares tools to manage architectural design decision and the ways used to model these decisions.

We selected 10 tools from these papers, the tools are: AEvol, Ontology-Driven Visualization (ODV), Archium, ADDSS, AREL, Knowledge Architect, PAKME, Web of Patterns, Stylebase for Eclipse, and Morpheus. We summarized the observations on these tools in Table 6.3, are we also identified some interesting facts related with this thesis for some of these tools:

- ODV [78]: this tool uses the ISO/IEC 9126 [117] to classify NFRs.
- AREL [203]: this tool takes in consideration NFRs as one of the elements of the architecture design. This tool helps in the design of the architecture using UML models and views.

Table 6.3: Comparison of AK tools

Ref.	Name	SADM	AK[1]	Platform	MDD	NFRs
[95]	AEvol	No	No	Eclipse	No	No
[78]	ODV	No	No	Windows desktop	No	Yes
[121]	Archium	No	Yes	Java/Compiler	No	Somehow
[55]	ADDSS	No	Yes	Web	No	Somehow
[203]	AREL	AREL	Yes	Enterprise Architect	No	Yes
[122]	Knowledge Architect	No	Yes	Excel plug-in	No	No
[4]	PAKME	No	Yes	Web/Java	No	Yes
[82]	Web of Patterns	No	Yes	Web/Eclipse	No	No
	Stylebase for Eclipse	QADA	Yes	Eclipse	Yes	Yes
[159]	Morpheus	ATRIUM	No	Windows desktop	Yes	Yes
[129]	RBML-PI	RBML	No	Windows desktop	Yes	Yes
	ArchiTech[2]	*Quark*	*Arteon*	*Eclipse plug-in*	*No*	*Yes*

[1] This column indicate if the tool is based on some AK conceptualization.
[2] Our AK tool, ArchiTech, is explained in Section 7.3.

- PAKME [4]: in this tool NFRs can be specified as keywords of architectural patterns that then can be reused for other projects. This tool is limited to textual knowledge.
- Stylebase for Eclipse[3]: this tool is a plug-in for Eclipse, that is capable to generate code for some architectural patterns, each pattern have a model associated (an image not a real model) and the principal NFRs that are improved (but in a very limited way).
- Morpheus [159]: this tool uses NFRs as constraints over functional requirements that then conditions the software architecture. It is presented as a MDD method that starts from requirements using goal oriented notations.

6.3.1 Analysis

First of all it is worth to remark that most of these tools are discontinued or created just a proof of concept. Also, one important fact is that all the tools that appear in this section are the result of an academic research (as far as we know, there is no software company offering similar products). If we look to the SADM and AK columns, we can see that most of the tools have

[3] `stylebase.tigris.org`

ways to manage the AK but only few have a well-defined method, this is not strange because most of them are oriented to document ADs but not to assist in the decision-making process. Finally, it is worth to mention that we did not find and explicit link between the AK conceptualizations mentioned in 6.1 and the tools mentioned in this section. The Table 6.3 summarizes the AK tools mentioned in this section.

Chapter 7

Architectural knowledge

Relation between the contents of this chapter and published papers.

Section 7.1 Main contributions of [22, 23].

Section 7.2 Main contributions of [23].

Section 7.3 Main contributions of [14, 15].

In the last decade, software architecture has become one of the most active research areas in software engineering. As a significant trend in this community, many researchers have stated that Architectural Decisions (ADs) are the core of software architecture [121, 206]. Under this view, software architecture has evolved from a structural representation to a decision-centered viewpoint [144]. In this scenario, manage and reuse of the knowledge related to ADs, the Architectural Knowledge (AK) [145], has gained the attention of many researchers. Ontologies are a consolidated way of representing knowledge, and have already been proposed for AK representation in the past (e.g., [1]). Aligning with this trend, we designed and implemented:

- Arteon, an ontology to manage and reuse AK.
- Quark, a method to assist software architects in architectural decision-making.

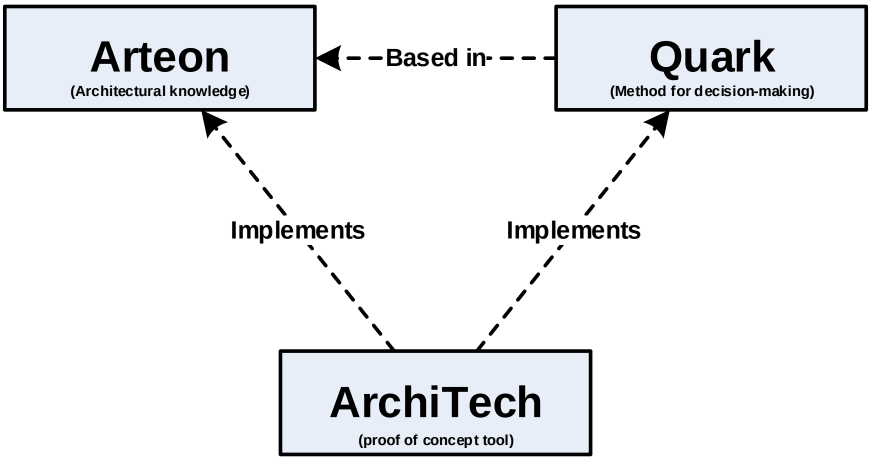


Figure 7.1: Arteon, Quark, and ArchiTech relations

- ArchiTech, a tool that acts as a proof of concept for both the ontology and the method.

Arteon organizes the AK, and keeps the decisions and their impact to quality attributes (QAs). Quark is built upon AK using the Arteon ontology as basis. ArchiTech is the implementation of both ideas in one single tool (Figure 7.1 depicts these relations). Some of the major highlights of Arteon, Quark, and ArchiTech are:

- They have been designed using empirical basis, as result of the empirical studies presented in Chapter 5.

- They use a decision-centered perspective, which is aligned with the actual trend in architectural research.

- They use NFRs to drive the decision-making.

7.1 Arteon: Architectural and Technological Ontology

The most important task of an architect is making Architectural Decisions (ADs) [121]. As we have seen in Chapter 5, in the current practice, architects made ADs based on their experience and intuition which may hamper understandability, traceability, and reuse of ADs, and this practice could be an important source of design mistakes. Architecture design is one of the first stages of the software development, where mistakes are translated into higher development costs, or into the worst scenario, a project failure. In practice, AK only resides in architects' minds because architects normally do not document their decisions, nor the reasoning and alternatives considered either.

One solution to this situation is to materialize AK, and as result we could benefit from it by sharing and reusing this knowledge in different software projects or inside a community of architects. We also want to use this materialized knowledge to guide and facilitate architects' the decision-making process, and, eventually, it could bring more reliability to the process by surfacing new alternatives that were not initially considered by the architect.

Among other alternatives we have chosen to use an ontology to materialize the AK. Ontologies have been successfully used for knowledge representation in other domains (e.g., software engineering, artificial intelligence, semantic web, biomedicine, etc.) and they offer other advantages such as reasoning and learning techniques ready to be applied (e.g., we could add new ADs using case-based reasoning techniques).

Sharing and reusing AK in different software projects and/or communities of architects are typical benefits of materializing this knowledge. But we also propose to use AK to guide and facilitate the architects' decision-making and, eventually, bring more reliability to this process by surfacing new alternatives that were not initially considered by the architect. To apply the learning and reasoning techniques necessary to walk this step, we need to be able to formalize this knowledge. In the next sections we provide some examples of formalizations of this knowledge to show its feasibility.

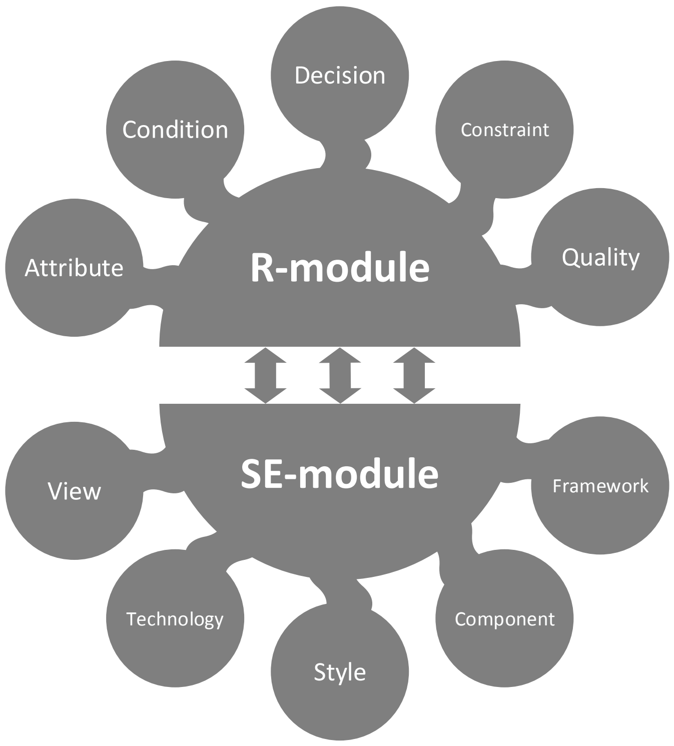


Figure 7.2: Arteon's overview

7.1.1 Design

Arteon is composed of two modules. The first one, the R-module, for reasoning and decision-making knowledge; and the other one, the SE-module, for managing structural elements, views and frameworks knowledge (see Figure 7.2):

Arteon's modules are interconnected, but we keep them loosely coupled and cohesive enough to be used or reused separately. The R-module and the SE-module are related through a specialization of the Decisional Element concept (explained later in this section) into a full classification of structural elements. With these two modules we are able to represent most of the knowledge related to architecture, the AK.

As complement to Arteon ontology, there could be another ontology that formalizes the software requirements knowledge (e.g., [123]). As we will see in the Section 7.2, we do not need an ontology for requirements because we assume that architects will transform the software requirements relevant for the architecture (e.g., requirements from a SRS expressed in natural language) into the constraints that can be handled by the Arteon ontology.

Arteon was designed following the principles stated by Gruber [102], Noy and Hafner [163], Guarino [105] and Evernmann [85]:

Clarity. *"An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective. [...] All definitions should be documented with natural language"* [102]. *"Be clear about the domain. Any formal theory is a theory about a domain. Such a domain must be clarified in advance"* [105]. We had many iterations in the design of the Arteon ontology, and we have resolved many ambiguities of the terms selected to represent the concepts that appear in the ontology.

Coherence. *"An ontology should be coherent: that is, it should sanction inferences that are consistent with the definitions. At the least, the defining axioms should be logically consistent"* [102]. The coherence and consistency of the ontology has been checked during its design by instantiating the concepts that appear in the ontology with toy examples of AK. This practice produced a faster evolution of the ontology design.

Extendibility. *"An ontology should be designed to anticipate the uses of the shared vocabulary. [...] One should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions"* [102]. Whenever possible, the definitions of Arteon's concepts are built upon the other concepts that appear in the ontology.

Reuse. *"In order to enable reuse as much as possible, ontologies should be small modules with high internal coherence and limited amount of interaction between the modules"* [163]. As mentioned before, Arteon is composed by two modules connected only by inheritance relationship.

Minimal encoding bias *"The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. An encoding bias results when a representation choices are made purely for the convenience of notation or implementation"* [102]. Arteon has been diagrammed using UML class diagrams to present and describe Arteon' concepts. UML has not been a limitation to express any concept or relationship. We also found in the literature many authors that use UML to diagram the ontologies (e.g., [106, 39]) and there is also the possibility to convert the UML representation of the ontology into OWL [96].

Minimal ontological commitment. *"An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities. An ontology should make as few claims as possible about the world being modeled"* [102]. Most of the concepts that appear in Arteon are adopted from the software architecture literature. They are defined carefully, and whenever possible we simply adhere to the most widely-accepted definition.

Identity. *"Identity criterion (and especially Lowe's principle, no individual can instantiate both of two sorts if they have different criteria of identity associated with them) can play a crucial role in clarifying ontological distinctions"* [105]. Since Arteon's generalizations are all disjoint we cannot incur in an identity issue.

Basic taxonomy. *"All ontologies are centered on a taxonomy, Such a taxonomy is the main backbone of the ontology, which can be fleshed with the addition of attributes and other relations among nodes. Isolate a basic taxonomic structure. Form a tree of mutually disjoint classes"* [105]. In Arteon this backbone taxonomy are the decisional elements, that are specialized in the SE-module.

Cognitive quality. *"An ontology, as a formal description of a domain, must conform to the way in which the domain is perceived and understood by a human observer"* [85]. We have tried to be as near as possible to the understanding of architects, to this end we used the experience earned from the interviews performed in the second empirical study (see Section 5.2).

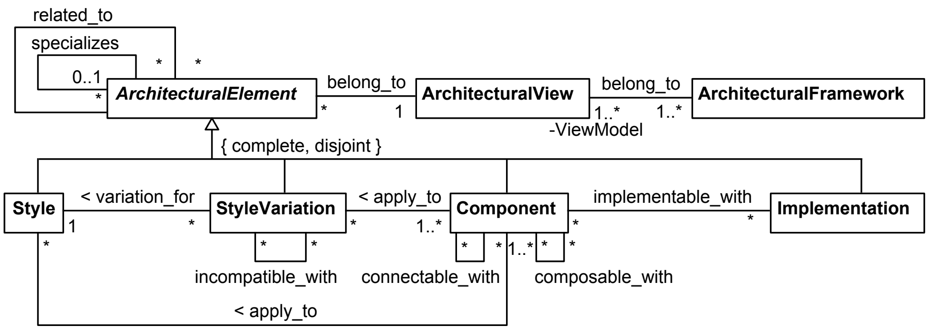


Figure 7.3: Arteon's SE-module

7.1.2 Structural elements module (SE-module)

In this section we present the SE-module of Arteon. In Figure 7.3 we show the concepts of this module and the relationships among them, whilst in Figure 7.4 we show an example of these concepts in a typical web-based application scenario. The concepts of the SE-module are enough to represent the structural representation of an architecture, which are the main elements that need to be considered during decision-making process carried by the architects. We describe next the most important concepts in the SE-module.

Architectural view

Representation of the whole system from the perspective of a related set of concerns [118]. Views are useful in large and complex architectures where trying to understand the whole architecture in a single representation could be, at least, a difficult task. In the example (Figure 7.4) there are 4 views: logical, development, deployment, and platform. Views can be used to show the static parts of the system, such as the ones in the example, or behavioral aspects, such as the process view defined in [141]. Our ontology can be used for both static and behavioral views, but our current work is more oriented to the static views.

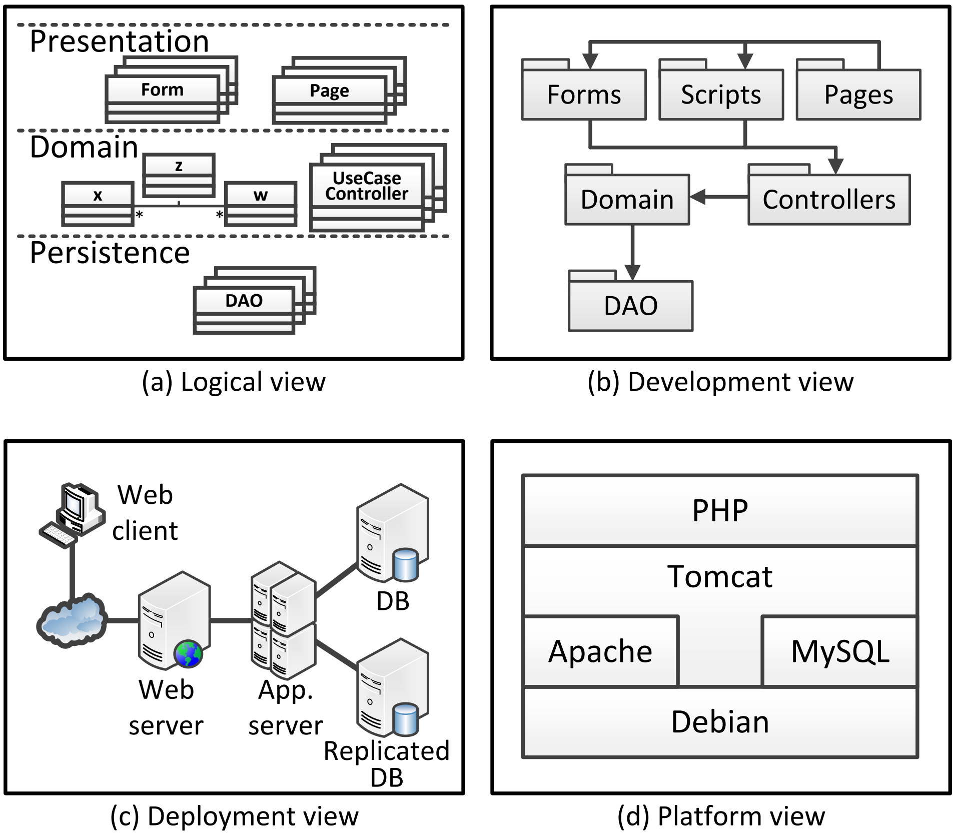


Figure 7.4: Example of the representable knowledge in SE-module

Architectural framework

Defines a set of views to represent the architecture, this set of views is also called *view model*. Examples of architectural frameworks are: RM-ODP [87] and *4+1* view model [141]. In the example (Figure 7.4) we use a variation of the *4+1* view model that takes into account the platform view. Other frameworks such as TOGAF [205] and Zachman [216] are partially supported because they define the full structure of an enterprise and we are only interested in the software part.

Architectural element

Abstract concept that denotes the kinds of elements that architects may decide to use in their architectural solutions. We consider four kinds of elements: styles, style variations, components, and implementations (see next definitions for particularities). All kinds of elements share some characteristics: they can be specialized (e.g., 3-layer style is a specialization of layered style). They can establish relationships or dependencies with other elements from other views. Looking at Figure 7.4 we can see some examples: Tomcat from the platform view is related to the application server, DAO classes are related to the DAO package, the scripts package is related to PHP, etc. Dependencies are especially useful to ensure the consistency of the architecture when a change is made.

Style

Architectural styles (or architectural patterns) were widely defined by [196] and [36]: *"An architectural pattern is determined by a set of element types, a topological layout of the elements indicating their interrelation-ships, a set of semantic constraints and a set of interaction mechanisms"*. Styles should not be confused with design patterns, styles define the whole structure of an architecture for a concrete architectural view, while a design pattern could be applied to one or more parts of the architecture (normally in the same architectural view). In the example: in the logical view we use a 3-layer style; in the development view we use a web application style; in the deployment view we use a specialized client-server style, database and application server separated [58]; and in the platform view we use a stack solution style.

Style variation

In practice, it is common that architectures that do not follow a *pure* architectural style. Instead, they follow a main style accompanied with some variations (examples of these variations for the layered style can be seen in [30]). Normally, the architect applies several variations (some of them are alternatives, see the incompatible relationship in Figure 7.3) to increase the satisfaction of the requirements. We can define a style variation as a minor style modification, e.g., a typical style variation is to apply a pattern in a concrete part of the architecture. In the example: the 3-layer style is modified with DAO and controllers patterns; the web deployment style is modified with a database replication; and the web platform style is modified with a FOSS variation. Currently we are not trying to deal with the complexity of using more than one style in one view, but in most cases one style accompanied with variations would suit.

Component

A component is a building block of an architectural view, examples could be: web server for the deployment view, layer for the logic view, or package for the development view. For each view, the style and the used variations will describe which components could be used and how the architecture is built. Components share two extra characteristics apart from the ones inherited from being architectural elements: first, components are connected to other components (e.g., presentation layer, that is a specialization of layer, is connected to the domain layer) and second, components can be composed by other components (e.g., layers are composed by modules).

Implementation

Implementations are the real pieces that will build the software architecture after its design. This part of the knowledge is becoming important as nowadays most of the software is built using existing pieces of software or, in some cases, hardware. In the example, the implementations would be: the classes implemented in some programming language, the package distribution provided by the programming language, the physical pieces of hardware where the system is deployed (e.g., a load balancer that is actually implemented by a device from Cisco Systems) and the concrete versions of the platform components. In the last two cases this knowledge could be reused in different architectures, and could be used to ensure the satisfaction of requirements or to detect incompatibilities. The non-reusable knowledge (e.g., implemented classes) would not be part of knowledge of this ontology.

To better understand the importance of this concept, we could think in Service Oriented Architectures (SOA). These architectures are composed of services that sometimes are already implemented by third-party companies. We can use the knowledge of the implemented services to design a SOA.

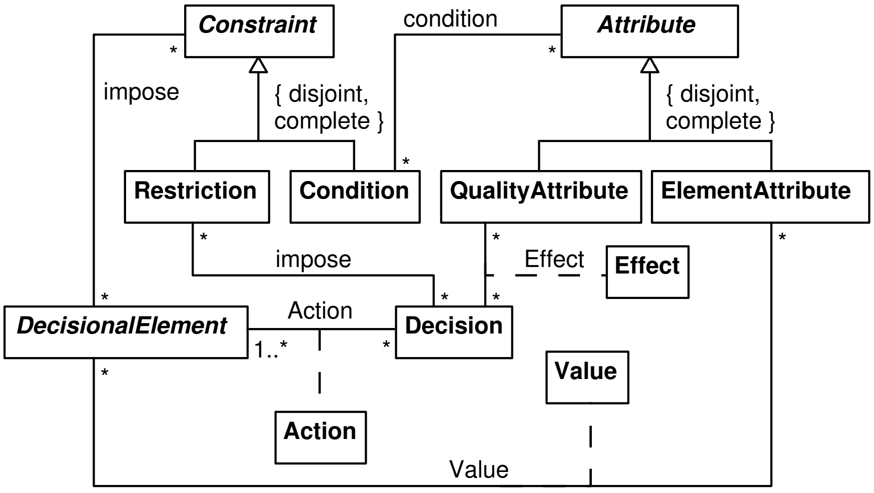


Figure 7.5: Arteon's R-module

7.1.3 Reasoning module (R-module)

In this section we present the R-module of Arteon. In Figure 7.5 we show the concepts of this module and the relationships among them. The concepts of the R-module allow representing architectural decisions, together with the rationale and the impact in software quality. Following we describe the most important concepts in the R-module.

Decisional element

A decisional element is an elemental part of the architecture that the architect can decide upon, i.e., the object of decisions. This concept is specialized in the previously seen SE-module as *Architectural element*, so it is left unrefined in the R-Module. The specializations in the SE-module are not the unique possible specialization of this concept, being a modular ontology makes it is easy to design and use a different specialization hierarchy for the decisional element (see *extendibility* ontology design principle).

Decision

According to RUP [140], software architecture is the *"selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization"*. This definition is about making ADs, structural and behavioral (i.e., existence decisions [145]).

In Arteon, the decision concept is very similar to the existence decision concept. Decisions are actions over decisional elements where the action determines the effect of the decision. Due to the extendibility design principle, we have not closed the ontology to a predefined set of actions, but possible actions could be, for example, the ones proposed in [145]: *use*, the decisional element will be in the architecture, and *ban*, the decisional element will be excluded from the architecture.

Constraint

Constraints can be imposed by software requirements or by decisional elements (note that the concept of requirement belongs to the Req-module of Arteon). Constraints coming from requirements are normally described in natural language (e.g., "the system shall be developed in C++"), sometimes using templates (e.g., Volere [183]) or a specialized language (e.g., temporal logic, the NFR Framework [63], etc.). Constraints coming from decisional elements are formalized as part of the AK (e.g., when the architect uses a technology that is only available for a particular platform, s/he is restricting the architecture to this platform).

Independently from the origin, we distinguish two kinds of constraints:

- *Restriction.* A constraint that directly imposes one or more ADs. For example, the "operating system" shall be "Debian".
- *Condition.* A constraint that specifies the valid values for attributes. E.g., if we only want to use Open-Source Software (OSS) software, the condition limit the "license" attribute to OSS licenses (e.g. GPL, LGPL, BSD).

In order to be able to reason with these constraints they must be formalized as evaluable expressions. Again, the ontology does not commit to

```
RestrictionSet → Restriction (LogicOp Restriction)*
Restriction → Action [DecisionalElement]
Action → <use> | <ban>
ConditionSet → Condition (LogicOp Condition)*
Condition → ComparativeCond | ConjuntiveCond
ComparativeCond → [Attribute] CompOp [Value]
ConjunctiveCond → [Attribute] ConjOp [Value]+
LogicOp → <and> | <or>
CompOp → <greater_than> | <lower_than> | <equal_to>
ConjOp → <includes> | <excludes>
```

Figure 7.6: CFG to formalize constraints

any particular proposal, but we provide an example expressed as a Context Free Grammar (CFG) [162] (see Figure 7.6). For simplification, we included extra notation in the CFG: *[concept]* means one valid instance of the concept and <*symbol*> means a terminal symbol. Also, for simplification, we did not include semantic rules (e.g., "the data type of the value should be the same of the data type of the attribute"). Depending on the expressiveness of the formalization, constraints could contain logic, comparative and conjunctive expressions, but expressiveness impacts negatively on the complexity of the reasoning system.

Attribute

An Attribute is an *"inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means"* [116]. In Arteon we differentiate two kinds of attributes:

- *Element Attribute.* An attribute of a Decisional Element. E.g., the values of the "license" attribute are the names of the licenses. Only Decisional Elements for which the license is relevant will have a value (e.g., technologies).

- *Quality Attribute.* An attribute that characterizes some aspect of the software quality. For example, ISO/IEC 25000 [116] defines a hierarchy of QAs (named "characteristics" in the standard: functionality, reliability, usability, efficiency, maintainability, and portability).

In this case, we also followed the extendibility principle by leaving the attributes customizable. Initially, we thought to propose a set of attributes, the most generic and independent of domain, but when we tried, we found out that domain-specific quality models may be more adequate in each situation (e.g., the S-Cube quality model [97] is specific for SOA) and that the element attributes are uncountable, and even worse, the same information can be modeled with different attributes (e.g., for the license information, we may have a boolean attribute, true when is a OSS license and false otherwise, or as before have an attribute with a list of licenses). We opted to let the domain expert decide which attributes are convenient in each case, but we acknowledge that more research is needed in order to make this knowledge reusable from one project to another.

7.1.4 Discussion

There are few works that use ontologies as the mechanism to represent the architectural knowledge. In particular for the structural elements part of the ontology we found: Akerman and Tyree [1], which has similar concepts to the structural elements presented in this work, but their ontology lacks of key concepts such as view and style. ArchVoc [31] had most of the concepts that appear in structural part of Arteon, but it does not have the conceptualization for architectural decisions. Pahl et al. [174] has the concepts related to architectural styles but uses the ontology with a different objective, as a modeling language. For more details see Section 6.1.

For the reasoning module of Arteon, it is worth to mention the Kruchten's ontology of ADs [145] proposes three kinds of decisions: existence decisions, property decisions, and executive decisions. In Arteon, Existence decisions, as mentioned before, are represented as the Decision concept and its actions. The two other kinds of decisions are also represented in the ontology, but not in an evident way. Property decisions are represented in Arteon as the resulting decisions from conditions over QAs or element attributes, for example, all the ADs made because of the condition to have OSS license. Executive decisions are represented in Arteon as the resulting ADs imposed by restrictions that come from the software requirements, in particular the requirements unrelated to the software quality, for example, a software requirement says that the DBMS should be Oracle, because the architect's company has a deal with Oracle to only use its products.

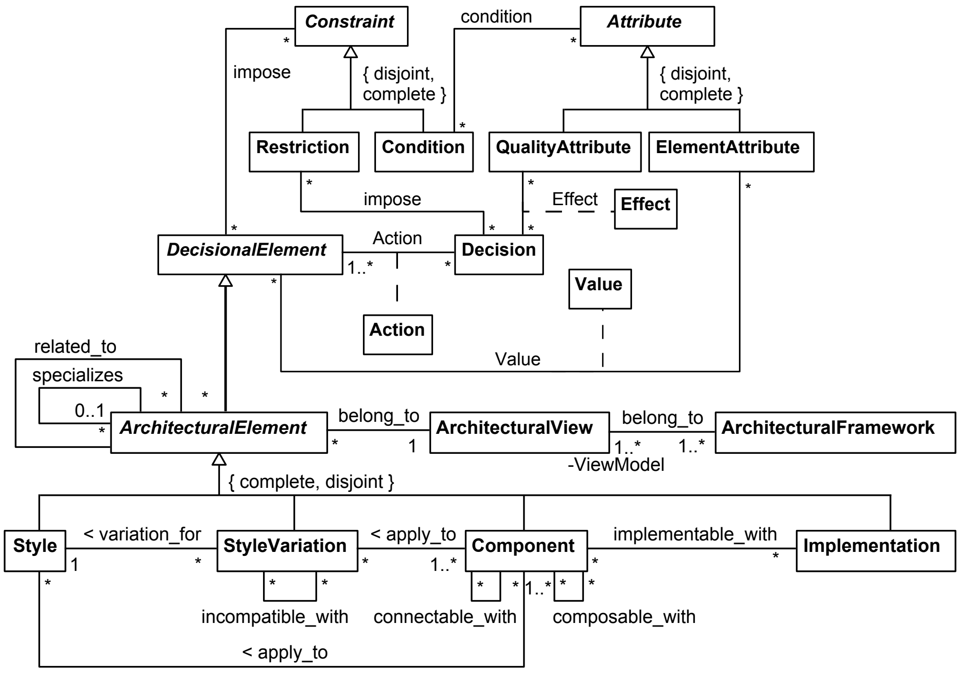


Figure 7.7: Arteon's SE and R modules together

In Figure 7.7 we can see the two modules together.

7.2 Quark: Quality in Architectural Knowledge

NFRs are among the principal drivers of the architectural decision-making process [62]. As mentioned in the introduction, it is not feasible to produce a software system that meets stakeholders' needs without taking NFRs into account.

Architects may have an idea of the impact of one particular Architectural Decision (AD) to the overall quality (i.e., what Quality Attributes (QAs) are improved or damaged by this AD), but it is hard to know if the architectural decisions made are respecting the NFRs. In the usual approach, architects use their own experience to produce software architectures that comply with the expected NFRs, but at the end, especially for crucial decisions, the architect has to deal with complex trade-offs analysis between QAs. This is even more complicated if we also consider the imposed constraints that include or exclude parts of the architecture (e.g., technologies). The architect, in addition to the NFRs trade-offs, will have to juggle with possible incompatibilities raised by the imposed constraints.

To alleviate this situation we present Quark, a method to assist software architects in architectural decision-making. The objective of Quark is to facilitate and making more reliable architects' decisions with regard to the desired qualities.

7.2.1 The Quark method

The design of Quark has been influenced by the observations gathered from the empirical studies presented in Chapter 5, the most important are:

(a) Software architects are the main source of NFRs. This is why the method is centered in the architect role.

(b) Software architects may be receptive to new design methods as far as they still keep the control on the final ADs. The method should suggest alternatives instead of making final ADs.

(c) The amount of information provided by the architects should pay itself. Software architects are pragmatic, a balance between effort and benefit must be reached in order to make the method suitable for them.

(d) The produced ADs should be justified, because architects also have to justify them to other stakeholders.

In Quark, the software architect plays the central role. Architects specify the NFRs and constraints (a). Architects select among the inferred ADs, and decide when the process has to end (b). In the same direction, Quark is not intrusive. It notifies about possible incompatibilities and possible actions to solve them, but the method does not require resolving any incompatibility to continue with the design, it is up to the architect (b). Using the Arteon ontology helps to reuse ADs (c) and also allows to produce detailed information on how an AD was reached, and why it was motivated (d).

The Quark method delivers an iterative process divided in four activities (see Figure 7.8): first, specification of the NFRs and the imposed constraints; second, inference of ADs; third, decision-making; and fourth, architectural

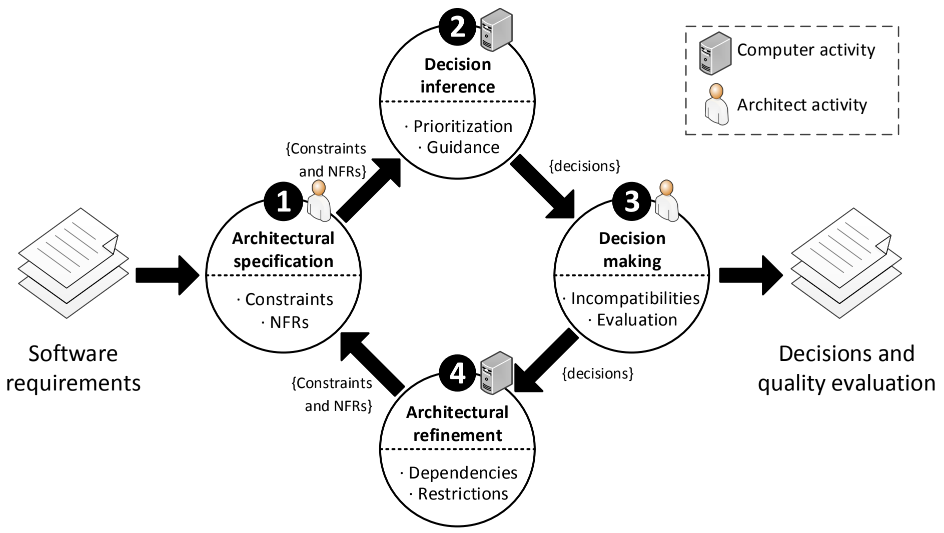


Figure 7.8: Quark overview

refinement (when necessary). Whenever the solution is refined, activities 1-3 are repeated. In the following subsections we give details on each activity.

Architectural Specification

In the first activity, the architect specifies the NFRs and constraints that are relevant for the architecture design. For example, a NFR could be "performance should be high" (in other words, more a goal than a requirement) or something more concrete as "loan processing response time should not be higher than two seconds 95% of the times". Constraints are typically referring to technologies, e.g., "the database management system (DBMS) must be MySQL 5", but may also refer to architectural principles, patterns or styles, as in "the architectural style must be Service-Oriented Architecture (SOA)". These requirements and constraints may come from the project documentation or from the architect's experience (as we found out in our empirical studies, see Chapter 5).

Due to Quark's iterative nature, and aligning with one observation of our empirical studies (the architect wants to have full control of the process), the specification of these NFRs and constraints does not need to be complete.

The architect has freedom to decide if s/he wants to start from a very short specification and then make the architecture grow in each refinement or if s/he wants to provide a more complete specification and see if the quality evaluation calculated by the method matches the expected NFRs, and then refine the architecture till it complies with the NFRs.

Decision Inference

In the second activity, the Quark method uses the AK available in the Arteon ontology (see Section 7.1) to generate a list of ADs. Since the expected amount of ADs in a real case is large, they should be prioritized using some criteria (e.g., the ADs that satisfy more constraints and better comply with the stated NFRs have higher priority).

ADs need to be informative. This means that, beyond their name, ADs must include information about: why the AD was offered?, what is the impact in the overall architecture quality?, and what other implications involve making the AD? (there are some works that offers more complete descriptions for ADs, for example, the template proposed in [206] or in [209]). For example, for the AD of using "data replication" we could answer the above questions as follows: "the AD of having data replication is offered because there is a NFR about having high performance", "by making this AD, the overall performance will increase but will affect negatively to the maintenance, and can damage the accuracy", "also, the used DBMS is required to be able to operate with data replication."

Decision-Making

In the third activity, the architect decides which ADs wants to make from the ones obtained in the previous activity. When the architect makes an AD, two things may happen. First, there could be incompatibilities with previous ADs (e.g., the architect decides to use "data replication", but s/he already selected a DBMS that does not support data replication), and second, there could be one or more NFRs that are not supported by the ADs made (e.g., the ADs made indicate that maintainability will be damaged while there is a NFR that says that maintainability is very important for this project). In both cases, the architect will be informed about which ADs are conflictive, but at the end s/he will decide if the set of ADs is satisfactory or not. In some cases there may be external reasons, not stated as NFRs or constraints, that have higher priority (e.g., the method recommends to use PostgreSQL but the development team is more experienced with MySQL, and there is not a big loss in the overall quality between both DBMS).

After the decision-making, the architect has the opportunity to conclude the process by accepting the current set of ADs or, alternatively, the architect may choose to start a new iteration of the full cycle. Here, as mentioned in [206], we understand the software architecture as a set of ADs.

Architectural Refinement

The Refinement activity is used for detecting issues that may be resolved in the next iteration. We identified three possible issues: incompatibilities, dependencies, and suggestions for NFRs.

- Incompatibilities (mentioned in the Decision-Making activity) are converted into new conditions over the attributes of the architectural elements defined in Arteon (e.g., the AD to use “data replication” sets a condition over an attribute “supports replication” for the “DBMS” architectural element).
- Dependencies occur when some AD requires other parts in the architecture (e.g., when the architect decides to use SOA, several related ADs are needed, service implementation: SOAP, REST, ...; service granularity: service composition, single service, ...; etc).
- Suggestions, we may infer that some QA is of special relevance due to the selected ADs (e.g., if many ADs have positive impact on security, Quark will suggest to the architect to include a NFR about security). This also helps making NFRs explicit.

The incompatibilities and dependencies are translated into constraints, while the suggestions imply new NFRs. As before, the architect has the last word, the architect will decide which of the new constraints and NFRs generated in the Refinement activity will be included in the Specification activity. At this point, the architect may also modify the constraints and NFRs (e.g., the architect may have noticed that one NFR is limiting the alternatives and decide to soften it).

7.2.2 Example

The present example focus on one AD, the selection of a DBMS, and one iteration in the Quark method. This example is mostly about technologies, but the same idea can be also applied, for example, to the selection of architectural patterns.

Following the Quark method, first the architect will identify the software requirements that are relevant to the architecture. For this example, the requirements are:

(R1) the software system shall keep the information about clients.

(R2) the software system shall be developed using OSS whenever possible.

(R3) the software system shall have backup systems for reliability.

Specification Activity

Once software requirements are identified, the software architect should translate them into NFRs and constraints. From R1, the architect may deduce that the project is an information system, so a DBMS will be necessary. R2 sets a constraint on the technologies used to have an OSS license. R3 sets constraints for backup facilities, and also mentions that reliability is a desired QA.

Using the formalization presented in Section 7.1, the specification is:

- *Use* DBMS
- "License" *includes* {"GPL", "LGPL", "BSD", etc.}
- "Backup facility" *equal* "yes"
- "Reliability" *greater than* "average"

Decision Inference Activity

Next, depending on the AK we have in the ontology and a prioritization criteria, an ordered list of ADs will be generated. For this example, the AK is based on the information published in the Postgres Online Journal [113] and the prioritization criteria is to give higher priority to ADs that satisfy more constraints and improve the selected QAs. The resulting list of ADs (with justifications) is:

1. The AD of using MySQL 5 is offered because it is OSS. There is no information available about backup facilities in MySQL. MySQL is preferred because it supports more OSS technologies. Using MySQL has neutral impact in reliability because ACID compliance depends on the configuration.

2. The AD of using PostgreSQL 8.3 is offered because it is OSS. There is no information available about back-up facilities in PostgreSQL. There are few OSS technologies with support for PostgreSQL. Using PostgreSQL improves reliability because it is ACID compliant.

3. The AD of using SQL Server 2005 is offered because it satisfies the backup facility condition. SQL Server is not OSS. There are few OSS technologies with support for SQL Server. SQL Server will require a Windows operating system. Using SQL server improves reliability because it is ACID compliant.

Decision-Making Activity

In the Decision-Making activity, the architect, for example, will decide to use MySQL 5 (the AD with higher priority) as the implementing technology for the DBMS component. But as said before in this paper, the architect may prefer to use PostgreSQL, even it is not the highest-ranked AD. What is important to notice here is that the architect is able to make informed ADs, and, eventually, new ADs that were unknown to her/him are taken into consideration.

Architectural Refinement Activity

After the Decision-Making activity the architectural design will continue with new iterations, where the AD of using MySQL may have some impact (e.g., in the selection of other technologies that are compatible with MySQL). This information will appear during the Refinement activity as dependencies and incompatibilities.

7.2.3 Discussion

There are many SADMs published in the literature, (see Section 6.2). Here we compare some of them with Quark:

- SEI methods, namely ADD for design and ATAM for analysis, are heavyweight methods that require large-scale projects to achieve a balance between what the method offers and the effort that supposes for the architects to use it. This balance is hard to achieve when projects are low- or medium-scale. In our case, we based Quark method in suggestions from architects working in low- or medium-scale projects, therefore we think that our method can be successfully applied to this kind of projects.
- QASAR [47]. As Quark, QASAR relies on NFRs, but in QASAR first there is a design based only on functional requirements, and then it is refined using the NFRs. Instead, Quark uses NFRs from the very beginning as the main driver of the decision-making.
- QADA [151]. As Quark, QADA is quality-driven and it is built upon a knowledge base, but it is not centered in ADs.
- AQUA [60]. AQUA uses a decision-centered approach as we do in Quark, the main differences between both methods are that AQUA does not have an ontological foundation and their method is not based on empirical studies. On the first hand, having an ontology to reason and mange knowledge is known as a good approach in many areas (e.g., artificial intelligence) but is true that currently it is not wide used in computer engineering research. On the second hand, having an empirical study on the main target community (software architects) helps to reduce the risk of having a solution disconnected from the real needs of this community.
- A. Bertolino et al. [44]. The main difference with Quark is that Bertolino's method does not deal with architectural decisions. Also, it is not clear what is the interaction with the software architect in the method presented by Bertolino et al.
- T. Al-Naeem et al. [2]. For the computation method, they rely on Multiple Attribute Decision Making (MADM), in particular Analytic

Hierarchy Process (AHP), to score each alternative decision. Quark method is based on artificial intelligence algorithms to score each alternative decision. We are not in a position to say which option is best, but they are clearly different.

- AREL [202]. They propose a conceptual model to manage this rationale, and the concern for software quality during the architecture design, but they do not describe which is the reasoning method used (if any), in this situation it is hard to compare their approach with Quark.
- ATRIUM [159]. Contrary to Quark, this method does not focus on decisions but in scenarios and requirements.
- L. Chung et al. [61]. Contrary to Quark, Chung's framework does not support to explicitly trade-off analysis between alternate design decisions.
- S. Bode et al. [45]. This approach is specialized only in security, while Quark could be used for any type of requirements (it all depends of the knowledge base provided).
- S. Kim et al. [129]. This method uses feature models to generate the architecture automatically. It is very similar to a product line for architectures. Product lines work for known and repetitive problems, but in Quark we leave the door open to customize the knowledge to any particular architectural area of interest (e.g., the architect may want to have many technological alternatives but does not care much about styles because s/he uses always the same).
- D. Perovich et al. [177]. There are also some differences, Perovich's method generates architectural models following the MDA approach, while Quark is more focused on the architectural decisions itself and the customization of the architectural knowledge.
- QRF [161]. In Quark the elicitation of requirements is performed previously, in fact, the architect is expected to only introduce the requirements that are architecturally relevant. To this end the QRF method could help in the identification of requirements relevant for the architectural design.

Comparing Quark to the Hofmeister et al. general model of architectural design methods [110] (also mentioned in Section 6.2) we can see that in Quark, the architectural analysis is covered by the Specification activity, the architectural synthesis is covered by the Decision Inference activity, and the architectural evaluation is covered by the Decision-Making activity, but there are two differences between Quark and the general approach of architectural design proposed by Hofmeister.

- The first difference is that Quark has an extra activity, the refinement, that facilitates the transition between iterations.

- The second difference is that Hofmeister's general approach deals with complete architectural solutions, while Quark works at decisional level. It is worth to remark that in our empirical studies we have detected that architects will not trust a support system that generates full architectural solutions without their intervention.

7.3 ArchiTech

ArchiTech is a tool developed as a proof of concept of the Arteon ontology and the Quark method[1]. The ArchiTech tool is capable to manage the AK as it is defined in the Arteon ontology (ArchiTech-CRUD[2]), and to assist architects in architectural decision-making as described in the Quark method (ArchiTech-DM).

Figure 7.9 shows an overview of the tool. We have defined two roles: the domain-expert, who will provide the AK using the ArchiTech-CRUD; and the architect, who will use the ArchiTech-DM to produce a software architecture with the help of ArchiTech.

7.3.1 ArchiTech-CRUD

This subsystem provides a graphical user interface for the domain expert to operate with the AK. The CRUD operations are specialized for four different types of knowledge:

[1]See the video at `www.upc.edu/gessi/architech/` for a running example

[2]CRUD stands for Create, Read, Update and Delete

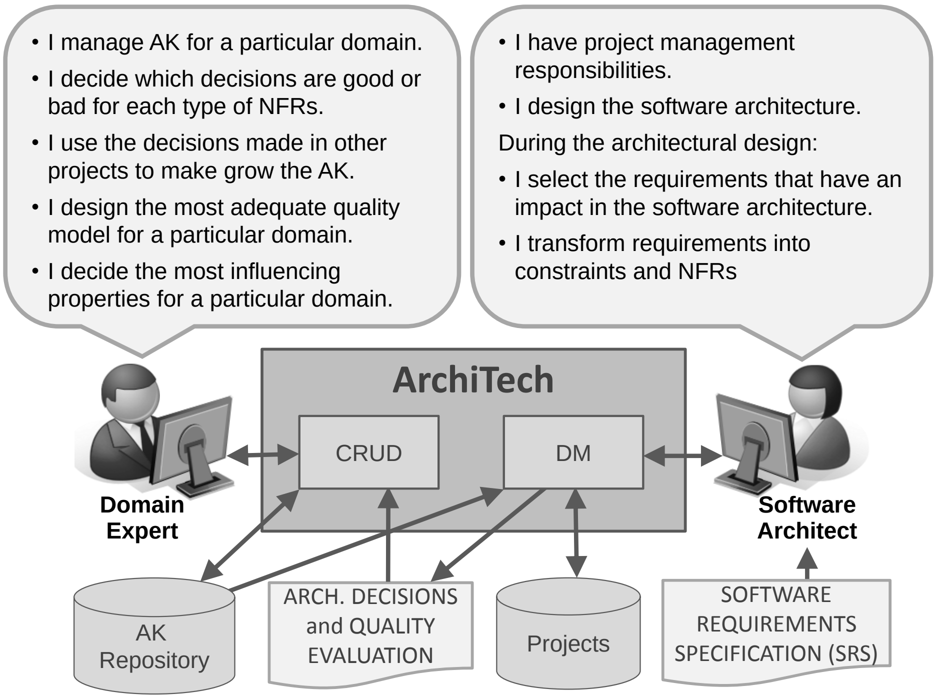

• I manage AK for a particular domain.
• I decide which decisions are good or bad for each type of NFRs.
• I use the decisions made in other projects to make grow the AK.
• I design the most adequate quality model for a particular domain.
• I decide the most influencing properties for a particular domain.
• I have project management responsibilities.
• I design the software architecture.
During the architectural design:
• I select the requirements that have an impact in the software architecture.
• I transform requirements into constraints and NFRs
ArchiTech
CRUD
DM
Domain Expert
Software Architect
AK Repository
ARCH. DECISIONS and QUALITY EVALUATION
Projects
SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Figure 7.9: ArchiTech overview

- *Architectural element.* The domain expert has to define the elements (e.g., architectural styles -SOA, layered, etc.-, components -services, packages, etc.-, technologies -DBMS, RESTful vs. W3C, etc.-) that are to be used to structure any software architecture. Contrary to Arteon, in Architect we have a fixed set of architectural views (logical, deployment, development, and platform) to classify these elements.

- *Element attributes.* Each architectural element may have values for one or more properties that are defined by the domain expert (e.g., the property License may be used to classify and reason about OSS technologies).

- *Quality attributes.* We give freedom to the domain expert to define (and reuse in multiple projects) the most appropriate quality model for her interests (e.g., a quality model to design SOA systems [97]).

- *Architectural decisions.* The domain expert has to define the decisions that are more habitual in a particular architectural domain (e.g., web-based system, service-based system, etc.), and which quality attributes are affected by each AD of the domain. Decisions can be higher-level (e.g., which architectural style to apply) or lower-level (e.g., which DBMS to choose).

In order to provide management facilities, the AK must be persistent and easy to share among projects. To this end we provide an embedded database, and we have also added an option to export the stored AK to an XML file.

7.3.2 ArchiTech-DM

This subsystem uses Quark method to guide software architects in quality-driven decision-making process. The four activities of the Quark method are implemented in ArchiTech as follows:

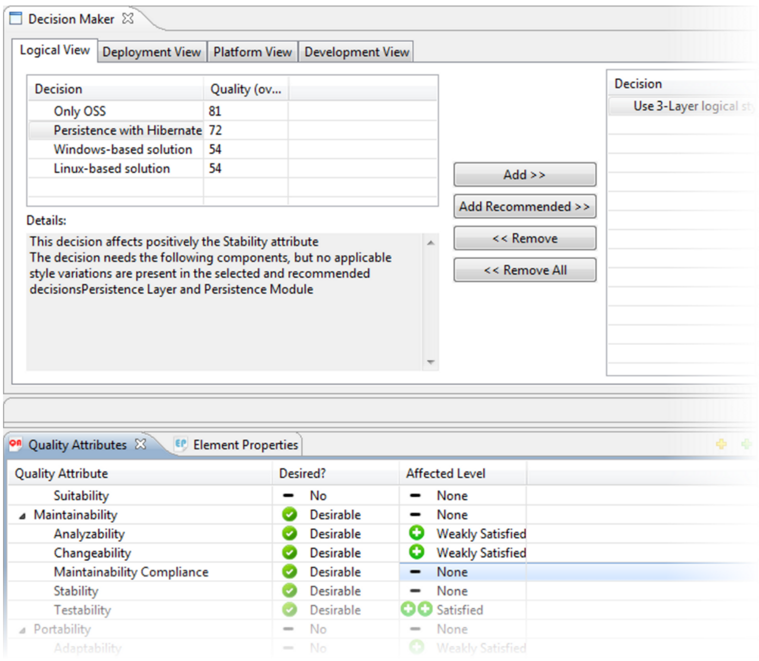
1. *Architectural Specification.* The architect can specify the NFRs and constraints in the user interface provided by the tool. The tool check them for correctness.

2. *Decision Inference.* The ArchiTech tool uses the AK defined in the CRUD part to generate a prioritized list of decisions using simulated annealing [130].

3. *Decision Making.* The architect decides what decisions are to be applied from the ones generated in the previous activity by selecting them from a list. Each decision has a description, and justification generated from the reasoning process (see the screenshot in Figure 7.10).

4. *Architectural Refinement.* The ArchiTech tool identifies possible issues and suggests actions to resolve them.

As it happens in Quark, after the fourth activity, we may end the process by accepting the resulting set of architectural decisions or use the suggested actions provided by the tool and start a new iteration. One extra feature, not mentioned in Quark, is that using this tool the architect can monitor the overall QAs evaluation while making ADs. This feature gives to the architect a clear notion of what is happening at any moment.



Decision	Quality (ov...
Only OSS	81
Persistence with Hibernate	72
Windows-based solution	54
Linux-based solution	54

Details:

This decision affects positively the Stability attribute
The decision needs the following components, but no applicable style variations are present in the selected and recommended decisionsPersistence Layer and Persistence Module

Decision
Use 3-Layer logical st

Quality Attribute	Desired?	Affected Level
Suitability	− No	− None
Maintainability	✓ Desirable	− None
Analyzability	✓ Desirable	+ Weakly Satisfied
Changeability	✓ Desirable	+ Weakly Satisfied
Maintainability Compliance	✓ Desirable	− None
Stability	✓ Desirable	− None
Testability	✓ Desirable	++ Satisfied
Portability	− No	− None
Adaptability	− No	+ Weakly Satisfied

Figure 7.10: ArchiTech screenshot

7.3.3 Design

We started this tool with a throw-away prototype, the experience was satisfactory since it demonstrated the feasibility of our approach, but for a more mature proof of concept, we discarded the prototype mainly because of two reasons: first, we wanted to integrate our tool with other MDD tools, following the ideas presented in the Part I of this thesis; and second, the AK was hard-coded, making it difficult to maintain or customize to different needs (e.g., different domains or projects).

- Concerning the first reason, we observed that most MDD tools are deployed as Eclipse plug-ins. Eclipse provides an open framework and a good community support which makes it a good candidate for software research communities. In order to present our solution to the MDD community, the use of Eclipse was a must for integration with other existing tools (e.g., code generators).

- For the second reason, in order to provide management facilities, the AK must be persistent. Due to the need of persistence, together with the need of providing a graphical interface to dialogue with the user and the existence of a domain logic (the CRUD use cases), we decided to adopt the classical Three-Layer architectural pattern.

Therefore, we finally end up with the situation of developing a three-layer architecture using a particular plug-in-based framework, Eclipse.

Development of a Three-layer architecture using plug-ins

To determine the alternatives for developing a three-layer logical architecture with plug-ins, we have to observe which are all the possible ways of grouping the logical components into the deployment components and compare those that make sense and are relevant. Within all the possibilities, a good starting point is to compare the two extreme alternatives since both make sense and have relevancy. The most intuitive one is to have a single plug-in that contains three packages, one for each layer (see Figure 7.11-a, three-layered plug-in). The second one is to separate each layer in an independent plug-in (see Figure 7.11-b, one plug-in per layer). The intermediate alternatives (e.g., the two top layers in one plug-in and the bottom layer in another plug-in) are not described because the trade-off analysis for these alternatives does not provide any remarkable (or unexpected) result.

The benefits and drawbacks of the Three-Layer architectural pattern are well-known and are the consequence of applying the principle described above. The separation into layers improves the reliability, reusability and portability of the application, while efficiency is the most damaged quality attribute because most of the calls have to go through all the layers. We refer to [51] for more details.

We consider three different perspectives of analysis for plug-in-based development coming from three different types of stakeholders that in our experience had been crucial to make the architectural decisions. The user, who installs the system in an execution environment and runs it; the developer, a general term representing software architects, programmers, etc., responsible to build and deploy the system; and the community, representing the set of potential users that are running similar systems or with a logical relation to the one of interest. The benefits and drawbacks of plug-in-based

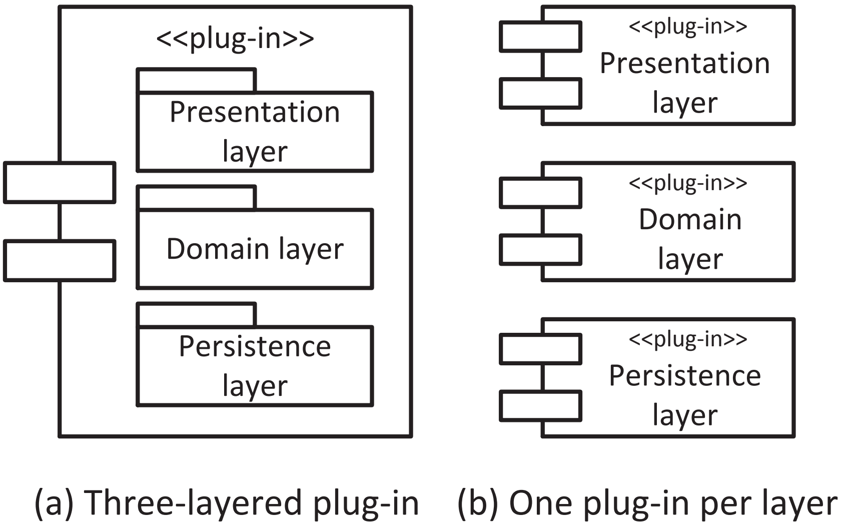


Figure 7.11: Three-layer architectural alternatives

development are:

- From the user's perspective, usability is the principal advantage; it implies that the user does not need to learn to use a brand new environment for the system functionality. But compatibility issues among different plug-ins hamper installability (when first installing the system) and reliability (since the system may stop delivering the promised functionality due to some pernicious interactions with new, incompatible plug-ins).

- From the developer's perspective, productivity is the main benefit, because the developer does not need to design a whole new application, instead s/he can just focus on the added functionality by reusing the functionality provided by the framework and other plug-ins, reducing thus the time and cost of the development (the time reduction is especially significant in the case that the developer is already familiar with the framework). But the adequacy of the chosen framework is

Table 7.1: Architecture alternatives comparison

	Benefits	Drawbacks
Three-layered plug-in	Maintainability and portability	Reusability and testability
One plug-in per layer	Changeability and adaptability	Reliability

very important, since plug-in development is restricted to the possibilities offered by that framework and it may be the case that it does not support adequately all the necessary functionality and/or technologies.

- From the community's perspective, interoperability between plug-ins is the main objective to achieve. For example, in many research communities all tools are developed using the same framework. But trends or fads can be a drawback, if a community decides to change the framework totally or partially (which is quite common with emergent technologies) the plug-in will become obsolete in a short time while a standalone application would have had a longer lifespan.

Table 7.1 summarizes the benefits and drawbacks of using three-layered plug-in or using one plug-in per layer.

The benefits of the second option (having one plug-in per layer) lead us to choose it, whilst being aware of the reliability-related drawback. One of the reasons for choosing this alternative is because splitting the three layers into independent plug-ins provides more flexibility. Once the decision was made, it was necessary to implement the solution. In the next section we report the differences between the expected theory as explored above and the real practice.

Dealing with architectural limitations imposed by technologies

The gap between theory and practice became relevant to our design when we started with the implementation of the plug-in and we began to find technological limitations.

As previously said, we decided that ArchiTech had to support a persistence mechanism to store AK in order to make it easily customizable by the user. As a result of this requirement, we decided to use some database for storing information. Since another requirement to satisfy was to have easy installation, we decided to provide ArchiTech with an embedded DBMS. The trade-off of this decision is that it limits cooperative work since every user will work with his or her own data. As a mitigation measure we decided to apply the Data Mapper design pattern [91]. With this decision, apart from other benefits coming from this pattern, we can exchange the embedded DBMS just by modifying a configuration file.

After some research we decided to use the following technologies in ArchiTech:

- *Eclipse framework* as the supporting framework for our plug-in due to community dominance. `www.eclipse.org/pde`
- *JFace* as the technology for the Presentation layer because it is mandatory in Eclipse plug-in development. `wiki.eclipse.org/JFace`
- *Hibernate* as the technology used to implement the Data Mapper. `www.hibernate.org`
- *H2* as the embedded DBMS, since it is an easy to use DBMS that is also well accepted by the Java developers community. `www.h2database.com`

At this point, we spent some time developing prototypes to test the feasibility of the architecture. During this activity we discovered two important technological limitations.

The libraries problem

During the development of the CRUD (Create-Read-Update-Delete) use cases of ArchiTech, we found that the Hibernate technology had an incompatibility issues with the Eclipse plug-in technology. After some research through some specialized sites we found out that the problem was related to the way in which the OSGi framework (which specifies how to deal with plug-ins, `www.osgi.org`) manages the loading of classes. The problem arises when a plug-in needs to invoke code from an external library that does not belong to the Eclipse framework. We solved this problem by embedding the libraries into a different plug-in.

Coupling between JFace and the Domain layer

JFace, being an integrated part of the Eclipse framework, is not affected by the previous problem but presents a different one. The adoption of JFace in the Presentation layer requires continuous synchronization between the user interface and the domain classes, which implies that the information shown in the user interface should be connected to the "real" objects of the Domain layer. Otherwise, the user interface may show inconsistent information. Despite of this fact, when retrieving objects from the database, in each subsequent call, Hibernate produce different instances of the same object.

To avoid this behavior, we had to implement a class to represent the current state of the domain objects, that is, references to all the objects that are being used directly or indirectly by the views, so that when an object is retrieved from the database, the references can be updated to point to it. The drawbacks of this solution come from the replication of information: less maintainability and less efficiency in data management. However, since ArchiTech does not manage huge amounts of data, efficiency loss is not significant.

ArchiTech final architecture

As result of these detected technological limitations, the theoretical architecture initially designed for ArchiTech had to be refactored into another one that is able to cope with these limitations. The resulting architectural solution is shown in Figure 7.12.

We had to split the persistence plug-in to solve the libraries problem, and we took the option to separate the libraries for Hibernate and the libraries for the H2 support, because making this extra separation facilitates the shift to another DBMS if decided in the future. Notice that these plug-ins only contain the libraries provided by Hibernate in one case, and the libraries needed to work with the DBMS in the other, but they do not contain any specific functionality.

The second change was to put the Domain layer together with the Persistence layer, the reason being that the way Hibernate works makes hard to maintain the Domain layer totally independent from the Persistence layer. Since we do not expect to change this technology in a near future, we opted for facilitating the development, but, as told before, the coupling between JFace and the Domain layer has been isolated in a class that maintains the references to the domain objects.

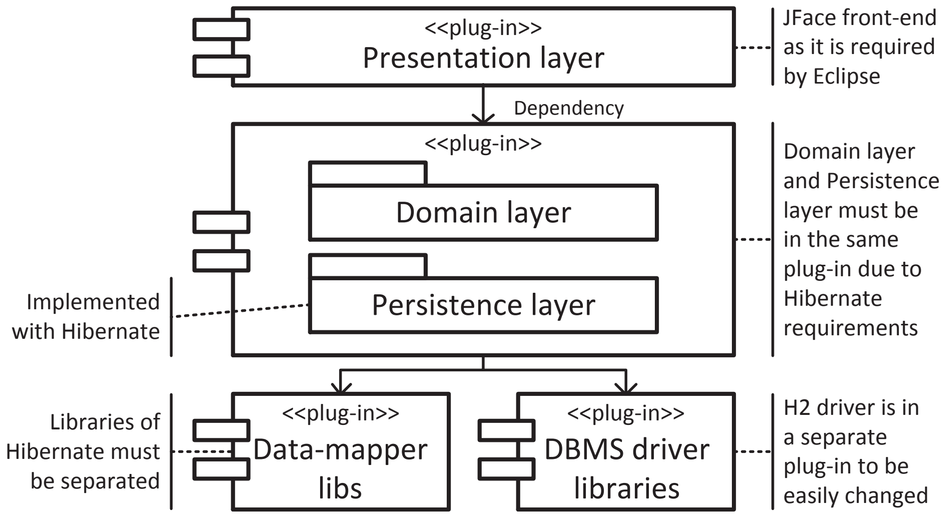

<<plug-in>>
Presentation layer
JFace front-end as it is required by Eclipse
Dependency
<<plug-in>>
Domain layer
Persistence layer
Domain layer and Persistence layer must be in the same plug-in due to Hibernate requirements
Implemented with Hibernate
<<plug-in>>
Data-mapper libs
<<plug-in>>
DBMS driver libraries
Libraries of Hibernate must be separated
H2 driver is in a separate plug-in to be easily changed

Figure 7.12: Architectural solution used in ArchiTech

7.4 Conclusions

One of the most known Kruchten's statements is "the life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in dark" [142]. The lack of knowledge is one of the reasons to produce suboptimal decisions. For example, the architect may not know all the effects of using some technology or architectural pattern: it may need of other components to work correctly (e.g., some of them may be incompatible with other ADs), it may have unexpected effects in the overall evaluation of some QAs (e.g., lowers the resource utilization efficiency). Also, the lack of knowledge may cause a worse situation when some alternative is not considered because it is unknown to the architect. To improve this situation we presented Arteon, an ontology to represent AK; Quark, a method to assist software architects in architectural decision-making; and we have described the principal parts of ArchiTech, and the design of this proof of concept tool.

We are aware about some of the limitations of the presented work, one of the major problems we have to deal with is the amount of knowledge required. Our position is that the best way to acquire and maintain such amount of knowledge is making architects active participants of its acquisition and maintenance. A possible way to achieve this participation is using networks of knowledge, which have been successful in other areas (e.g., Stack Overflow for software developers). Other techniques that have been considered to acquire and maintain this knowledge are knowledge reuse and knowledge learning, but both have drawbacks, for example, reusing knowledge you may find out that a solution that provides high security in a information system may not be secure enough for a critical system, and in order to use learning techniques first is necessary to have a big source of knowledge.

More information about ArchiTech may be found at (`www.upc.edu/gessi/architech/index.html`), where the current version is available for download.

Conclusions and future work

In this chapter we present the conclusions of this thesis by answering the initial research questions, and then we discuss the possible future work from the current state of the research.

Conclusions

In the introduction of this thesis, we stated three research questions. In this section we provide answers to these questions.

RQ1: *How NFRs and architecture can be integrated in the MDD process?*

Answer: the answer to this question has been the design of a theoretical approach that handles NFRs in the MDD process. We explored different variants of the proposed approach and we compared it to the current state of practice. As a consequence of this integration we detected the need to include architectural design as part of the MDD process. We drafted a viable path to include NFRs into MDD, but due to more fundamental questions regarding to the relation between NFRs and ADs, this part of the work had to stop at the theoretical level.

RQ2: *How do NFRs impact on architectural design?*

Answer: our approach to face this research question has been the design and execution of several empirical studies oriented to understand the way of thinking of software architects. These studies confirmed some of the common beliefs about software architecture (e.g., that the role of NFRs is fundamental in the architecture design) and uncovered some facts that were not known, or at least not evident (e.g., most architects play many roles in the software companies; architects are the main source of NFRs instead of the client). These studies have also been of much value to understand architect's needs, which were very useful in the design of the Arteon ontology and the Quark method.

RQ3: *Which AK is necessary to make architectural decisions?*

Answer: from the experience obtained both from the architects, in the empirical studies, and from the academics, in the literature reviews, we were able to design an ontology that covers a great part of the architectural decisions, in particular the ones related to the structure of the architecture. To prove the feasibility of this ontology we designed a method to assist software architects in the architectural decision-making process (again, relaying on the feedback obtained from the architects themselves) and a tool were we can actually add, modify, and create architectural knowledge, and then use it to make architectural decisions.

As a final conclusion, the most valuable outcome of this thesis as a whole is the exploration of different perspectives of the role of NFRs in the software architecture design: in the first part of this thesis, we proposed a way to integrate NFRs in MDD, which made evident the need to support architecture design in MDD; in the second part, we observed how NFRs are understood by architects, and how NFRs are used in practice; and finally, in the third part, we designed Arteon and Quark, an ontology and its companion method, where NFRs are used to drive the architectural decision-making. On the whole, the thesis has served as a way to improve the understanding and the knowledge related to the role of NFRs in software architecture design.

Future work

We could divide the future work in three parts, corresponding to the three parts of this thesis:

- For the first part, we are interested in collaborating with other research groups experienced in MDD to help in integrating NFRs in their MDD approaches. Now that we have better notion of the role of NFRs in software architecture design, we want to produce an implementation of the proposed framework to integrate NFRs in MDD. A possible way to reach this objective is to strategically propose final career projects that together will make the whole implementation.
- For the second part, we are interested in contrasting the results obtained in the empirical studies. We have planned collaborations with other researchers in the field to compare our results. For the third empirical study we are planing to produce an extended version with more responses.
- For the third part, we are interested in continuing the research line about how to better integrate NFRs in architectural decision-making process in a way that improves the overall quality of the produced architectures. We currently have a proof of concept tool and we need to use it in experiments and in real practice to obtain feedback that could be used to improve both the Arteon ontology and the Quark method.

Another line of exploration is to apply the experience and knowledge obtained in this thesis to a particular domain, for example, the Open Source Software (OSS). We could apply methods for decision-making in this particular domain and execute empirical studies to understand fundamental differences between the OSS and closed software with regard to software architects and software architecture design.

Appendix A

First empirical study

Questionnaire of the first empirical study:

Model-Driven Development in IT companies and organizations

Model-Driven Software Development in IT companies and organizations

With this questionnaire we analyze the usage of Model-Driven Software Development in IT companies and organizations. We are mainly interested to know about **experiences** on Model-Driven Software Development initiatives, the **desirable automation level** in the software development process and the **importance of non-functional requirements** in this process.
Model-Driven Software Development (MDSD) is based on the construction of a system model that can be transformed, in a systematic and semiautomatic way, into an implementation deployed on one or more software platform technologies. The system model can be unique or can be a combination of models (e.g., UML models). The concept of MDSD is also known with other names that are basically similar: MDA: Model-Driven Architecture; MDD: Model-Driven Development; MDE: Model-Driven Engineering, etc.

This questionnaire is **anonymous** and it will take you about **15 and 20 minutes.**

There are 50 questions in this survey

Personal data

1 [DP1]Name (optional)

Please write your answer here:

2 [DP2]Company or organization (optional)

Please write your answer here:

3 [DP3]E-Mail (if you wish to receive the results)

Please write your answer here:

4 [DP4]Current position in the company or organization *

Please write your answer here:

5 [DP5]Education to date related to software development *

Please write your answer here:

Generic development of software projects

Note: Answer this group of questions **without taking into account** whether the projects were made using MDSD or not.

6 [Arch]Choose the architectural styles used in your projects: *

Please choose **all** that apply:

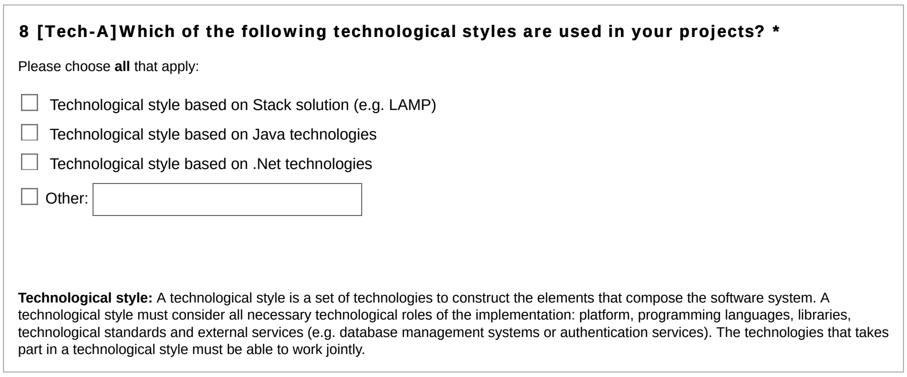
- ☐ Service-Oriented Architecture (SOA)
- ☐ 3-layered Architecture
- ☐ Client-Server Architecture
- ☐ Peer-to-peer Architecture
- ☐ Database-centric Architecture
- ☐ Event-Driven Architecture
- ☐ Component-based Architectures (plugins, add-ons, extensions, components)
- ☐ Pipe and filter Architecture
- ☐ Mainframe Architecture
- ☐ Model, View, Controler (MVC)
- ☐ Other: __________

Architectural style: We understand architectural style as the collection of the main elements that compose the software system and the strategy of communication used between them. Examples of software architectures are: 3-layered architecture, service oriented architecture, client-server, etc. A software system can be designed as a composition of many architectural styles depending on its needs.

7 [Tech]Choose the type of software developed in your projects: *

Please choose **all** that apply:

- ☐ Web services
- ☐ Web applications
- ☐ Distributed applications based on components
- ☐ Desktop applications
- ☐ Software for mobile devices
- ☐ Software for embedded systems
- ☐ Host applications
- ☐ Other: __________



8 [Tech-A]Which of the following technological styles are used in your projects? *

Please choose **all** that apply:

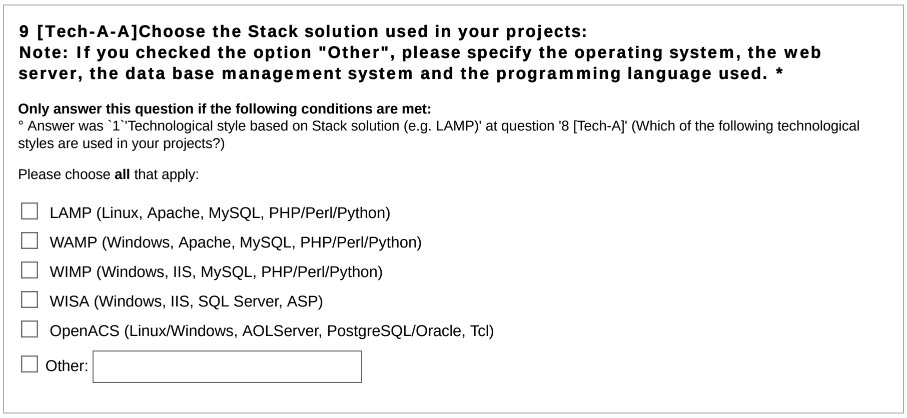
☐ Technological style based on Stack solution (e.g. LAMP)

☐ Technological style based on Java technologies

☐ Technological style based on .Net technologies

☐ Other:

Technological style: A technological style is a set of technologies to construct the elements that compose the software system. A technological style must consider all necessary technological roles of the implementation: platform, programming languages, libraries, technological standards and external services (e.g. database management systems or authentication services). The technologies that takes part in a technological style must be able to work jointly.



9 [Tech-A-A]Choose the Stack solution used in your projects:
Note: If you checked the option "Other", please specify the operating system, the web server, the data base management system and the programming language used. *

Only answer this question if the following conditions are met:
° Answer was `1`'Technological style based on Stack solution (e.g. LAMP)' at question '8 [Tech-A]' (Which of the following technological styles are used in your projects?)

Please choose **all** that apply:

☐ LAMP (Linux, Apache, MySQL, PHP/Perl/Python)

☐ WAMP (Windows, Apache, MySQL, PHP/Perl/Python)

☐ WIMP (Windows, IIS, MySQL, PHP/Perl/Python)

☐ WISA (Windows, IIS, SQL Server, ASP)

☐ OpenACS (Linux/Windows, AOLServer, PostgreSQL/Oracle, Tcl)

☐ Other:

10 [Tech-A-B]Choose the Java technologies used in your projects: *

Only answer this question if the following conditions are met:
° Answer was `2`'Technological style based on Java technologies ' at question '8 [Tech-A]' (Which of the following technological styles are used in your projects?)

Please choose **all** that apply:

☐ Struts
☐ Spring
☐ JPA/Hibernate
☐ SEAM
☐ EJB 2
☐ EJB 3
☐ JAX-WS
☐ JAX-RPC
☐ Java Server Faces (JSF)
☐ Java Server Pages (JSP)
☐ Java Servlets
☐ Other:

11 [Tech-A-C]Choose the .Net technologies used in your projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'Technological style based on .Net technologies' at question '8 [Tech-A]' (Which of the following technological styles are used in your projects?)

Please choose **all** that apply:

☐ ADO.Net
☐ ASP.Net
☐ WCF - Windows Communication Foundation
☐ WF - Windows Workflow Foundation
☐ WPF - Windows Presentation Foundation
☐ Spring.Net
☐ NHibernate
☐ Windows Forms
☐ Other:

12 [DBMS]Choose the type of data base used in your projects: *

Please choose **all** that apply:

- ☐ Relational
- ☐ Multidimensional
- ☐ Object-Relational
- ☐ Object-Oriented
- ☐ Documental
- ☐ Deductive
- ☐ XML
- ☐ Other: ______

13 [DBMS-A]Choose the Data Base Management System (DBMS) used in your projects: *

Please choose **all** that apply:

- ☐ MySQL
- ☐ PostgreSQL
- ☐ Oracle
- ☐ SQL-Server
- ☐ DB2
- ☐ Other: ______

14 [DBMS-B]Choose the relevance of the following DBMS capabilities in your projects:

Please choose the appropriate response for each item:

	None	Marginal	Medium	Important	Critical
Stored procedures	○	○	○	○	○
Triggers	○	○	○	○	○
Schema validation (e.g. checks)	○	○	○	○	○

15 [NFR]Which of the following statements better describes the importance of non-functional requirements to you? *

Please choose **only one** of the following:

- ○ I don't consider them, I focus on the functional part
- ○ I consider them but I don't use them to take important decisions
- ○ They have the same importance as functional requirements

Functional requirement: Functional requirements establish the observable behavior that must exhibit the system (calculations, manipulations, listings, evolution aspects, etc.), as well as the data types specification.

Non-functional requirement: Non-functional requirements establish the criteria or global qualities of the software system and set restrictions (internal and external) on the software and the development process. Common types of non-functional requirements are: usability, efficiency and portability.

16 [NFR-X]Do you use the non-functional requirements to choose between different architectural styles and / or technological styles? *

Only answer this question if the following conditions are met:
° Answer was `3`'They have the same importance as functional requirements' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?)

Please choose **only one** of the following:

○ Yes
○ No

17 [NFR-A]Choose the relevance of the following types of non-functional requirements on the development of your software projects:

Only answer this question if the following conditions are met:
° Answer was `2`'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?)

Please choose the appropriate response for each item:

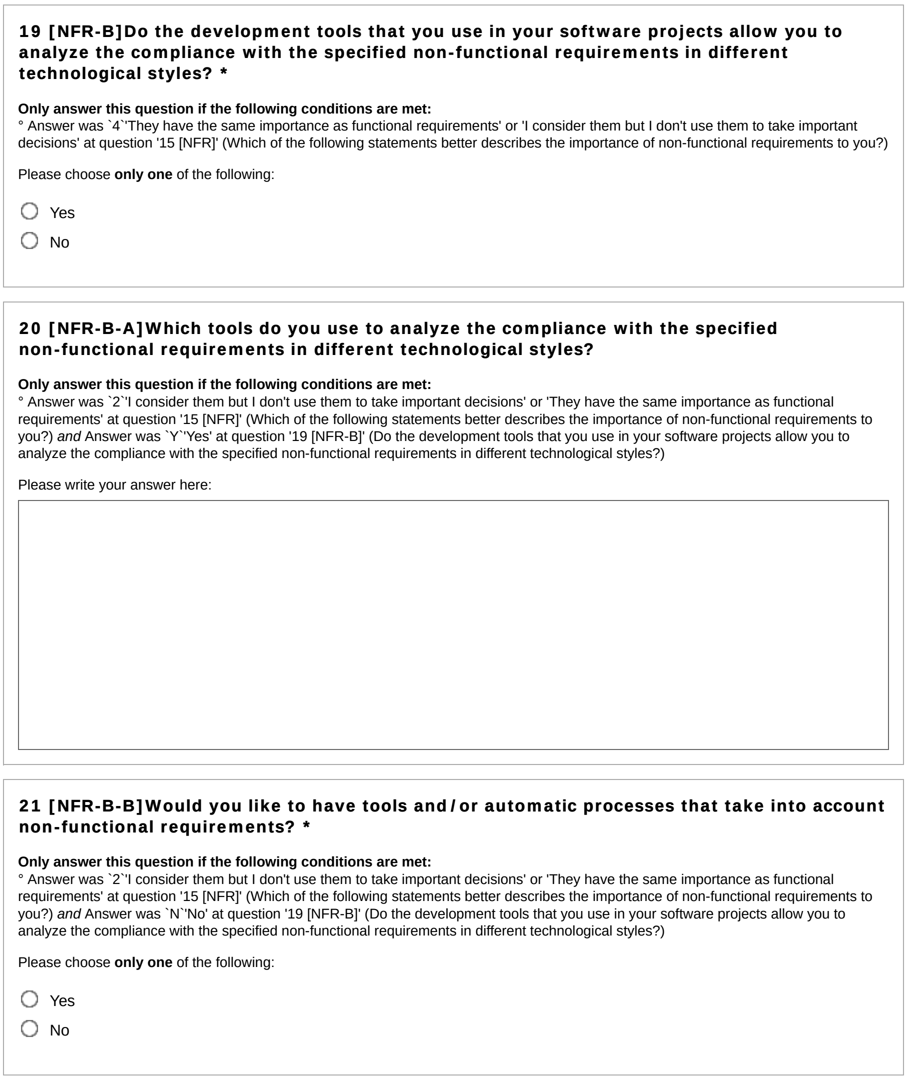
	None	Marginal	Medium	Important	Critical
Maintainability	○	○	○	○	○
Reusability	○	○	○	○	○
Efficiency	○	○	○	○	○
Reliability	○	○	○	○	○
Usability	○	○	○	○	○
Portability	○	○	○	○	○
Cost	○	○	○	○	○
Standards compliance	○	○	○	○	○
Organizational	○	○	○	○	○

Organizational requirements refer to aspects of the organization where the software system will be deployed.

18 [NFR-A-A]Do you consider other non-functional requirements during the development of your software projects?

Only answer this question if the following conditions are met:
° Answer was `2`'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?)

Please write your answer here:



19 [NFR-B]Do the development tools that you use in your software projects allow you to analyze the compliance with the specified non-functional requirements in different technological styles? *

Only answer this question if the following conditions are met:
° Answer was `4`'They have the same importance as functional requirements' or 'I consider them but I don't use them to take important decisions' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?)

Please choose **only one** of the following:

○ Yes
○ No

20 [NFR-B-A]Which tools do you use to analyze the compliance with the specified non-functional requirements in different technological styles?

Only answer this question if the following conditions are met:
° Answer was `2`'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?) *and* Answer was `Y`'Yes' at question '19 [NFR-B]' (Do the development tools that you use in your software projects allow you to analyze the compliance with the specified non-functional requirements in different technological styles?)

Please write your answer here:

21 [NFR-B-B]Would you like to have tools and/or automatic processes that take into account non-functional requirements? *

Only answer this question if the following conditions are met:
° Answer was `2`'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' at question '15 [NFR]' (Which of the following statements better describes the importance of non-functional requirements to you?) *and* Answer was `N`'No' at question '19 [NFR-B]' (Do the development tools that you use in your software projects allow you to analyze the compliance with the specified non-functional requirements in different technological styles?)

Please choose **only one** of the following:

○ Yes
○ No

Interaction level

22 [NI-Impl]For the following tasks of the implementation phase, choose the interaction level that you consider more adequate assuming that an hypothetic support tool is available. (1 to 5 as shown)

1: I wouldn't use any supporting tool to perform this task
2: The hypothetic support tool should ask me before taking any decision
3: The hypothetic support tool should ask me only before taking the relevant decisions
4: The hypothetic support tool should take the decisions for me but later I would check them
5: The hypothetic support tool would take the decisions for me without further confirmation
*****

Please choose the appropriate response for each item:

	1	2	3	4	5
Generation of the skeleton code	○	○	○	○	○
Generation of the code for a specific technology	○	○	○	○	○

23 [NI-Dsgn]For the following tasks of the design phase indicate the interaction level that you consider more adequate assuming that an hypothetic support tool is available. (1 to 5 as shown in the previous question) *

Only answer this question if the following conditions are met:
° Answer was NOT `N`'No' at question '21 [NFR-B-B]' (Would you like to have tools and/or automatic processes that take into account non-functional requirements?)

Please choose the appropriate response for each item:

	1	2	3	4	5
Selection of the architectural style that better conforms to the non-functional requirements of the software system	○	○	○	○	○
Selection of the technological style that better conforms to the non-functional requirements of the software system	○	○	○	○	○

Functional requirement: Functional requirements establish the observable behavior that must exhibit the system (calculations, manipulations, listings, evolution aspects, etc.), as well as the data types specification.

Non-functional requirement: Non-functional requirements establish the criteria or global qualities of the software system and set restrictions (internal and external) on the software and the development process. Common types of non-functional requirements are: usability, efficiency and portability.

Architectural style: We understand architectural style as the collection of the main elements that compose the software system and the strategy of communication used between them. Examples of software architectures are: 3-layered architecture, service oriented architecture, client-server, etc. A software system can be designed as a composition of many architectural styles depending on its needs.

Technological style: A technological style is a set of technologies to construct the elements that compose the software system. A technological style must consider all necessary technological roles of the implementation: platform, programming languages, libraries, technological standards and external services (e.g. database management systems or authentication services). The technologies that takes part in a technological style must be able to work jointly.

Model-Driven Software Development (MDSD)

24 [DSDM]According to your knowledge and skills, which of the following categories do you belong to? *

Please choose **only one** of the following:

○ I don't know what is Model-Driven Software Development
○ I know the concept of Model-Driven Software Development but I don't use it in my work
○ I have used the Model-Driven Software Development paradigm in my work

25 [DSDM-A1]Choose the initiatives you know: *

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

☐ Model-Driven Architecture (MDA)
☐ Model-Driven Development (MDD/MDSD)
☐ Model-Driven Engineering (MDE)
☐ Other:

26 [DSDM-A2]Choose the Model-Driven Development platforms you know: *

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

☐ Eclipse EMP
☐ AndroMDA
☐ openArchitectureWare
☐ I don't know any
☐ Other:

27 [DSDM-A3]Which Model Driven Software Development CASE tools do you know? (Model editors, etc.)

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please write your answer here:

28 [DSDM-B1]In how many projects have you applied Model Driven Software Development? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

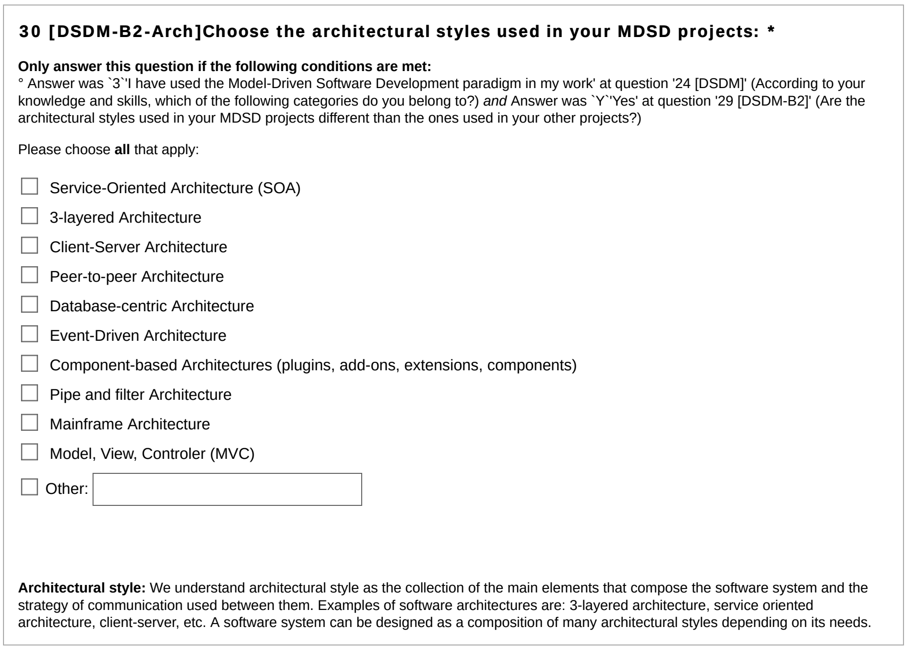
Please write your answer here:

29 [DSDM-B2]Are the architectural styles used in your MDSD projects different than the ones used in your other projects? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **only one** of the following:

- ○ Yes
- ○ No



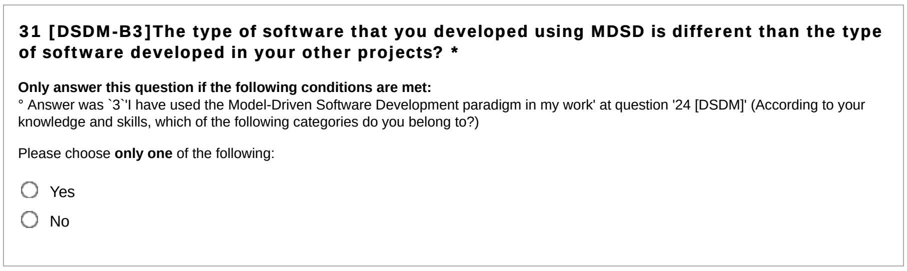
30 [DSDM-B2-Arch]Choose the architectural styles used in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '29 [DSDM-B2]' (Are the architectural styles used in your MDSD projects different than the ones used in your other projects?)

Please choose **all** that apply:

☐ Service-Oriented Architecture (SOA)
☐ 3-layered Architecture
☐ Client-Server Architecture
☐ Peer-to-peer Architecture
☐ Database-centric Architecture
☐ Event-Driven Architecture
☐ Component-based Architectures (plugins, add-ons, extensions, components)
☐ Pipe and filter Architecture
☐ Mainframe Architecture
☐ Model, View, Controler (MVC)
☐ Other:

Architectural style: We understand architectural style as the collection of the main elements that compose the software system and the strategy of communication used between them. Examples of software architectures are: 3-layered architecture, service oriented architecture, client-server, etc. A software system can be designed as a composition of many architectural styles depending on its needs.



31 [DSDM-B3]The type of software that you developed using MDSD is different than the type of software developed in your other projects? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **only one** of the following:

○ Yes
○ No

32 [DSDM-B3-Tech]Choose the type of software developed in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '31 [DSDM-B3]' (The type of software that you developed using MDSD is different than the type of software developed in your other projects?)

Please choose **all** that apply:

☐ Web services
☐ Web applications
☐ Distribuidas applications based on components
☐ Desktop applications
☐ Software for mobile devices
☐ Software for embedded systems
☐ Host applications
☐ Other: ____________

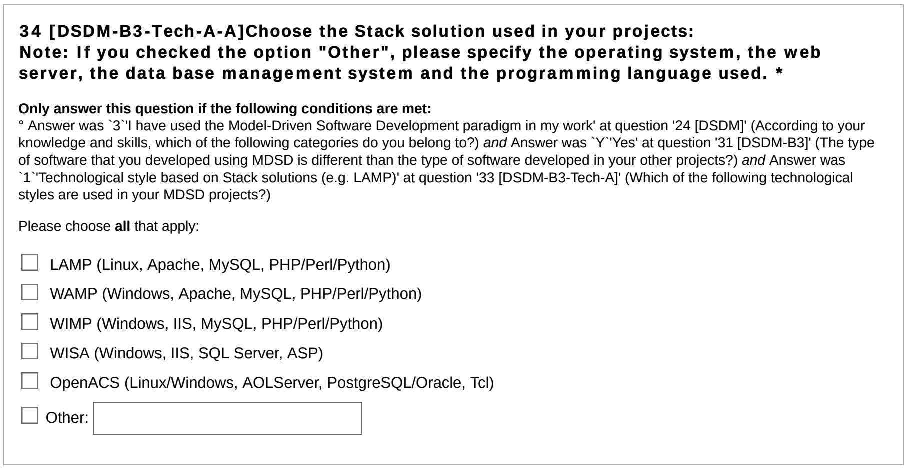
33 [DSDM-B3-Tech-A]Which of the following technological styles are used in your MDSD projects? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '31 [DSDM-B3]' (The type of software that you developed using MDSD is different than the type of software developed in your other projects?)

Please choose **all** that apply:

☐ Technological style based on Stack solutions (e.g. LAMP)
☐ Technological style based on Java technologies
☐ Technological style based on .Net technologies
☐ Other: ____________

Technological style: A technological style is a set of technologies to construct the elements that compose the software system. A technological style must consider all necessary technological roles of the implementation: platform, programming languages, libraries, technological standards and external services (e.g. database management systems or authentication services). The technologies that takes part in a technological style must be able to work jointly.



34 [DSDM-B3-Tech-A-A]Choose the Stack solution used in your projects:
Note: If you checked the option "Other", please specify the operating system, the web server, the data base management system and the programming language used. *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '31 [DSDM-B3]' (The type of software that you developed using MDSD is different than the type of software developed in your other projects?) *and* Answer was `1`'Technological style based on Stack solutions (e.g. LAMP)' at question '33 [DSDM-B3-Tech-A]' (Which of the following technological styles are used in your MDSD projects?)

Please choose **all** that apply:

☐ LAMP (Linux, Apache, MySQL, PHP/Perl/Python)

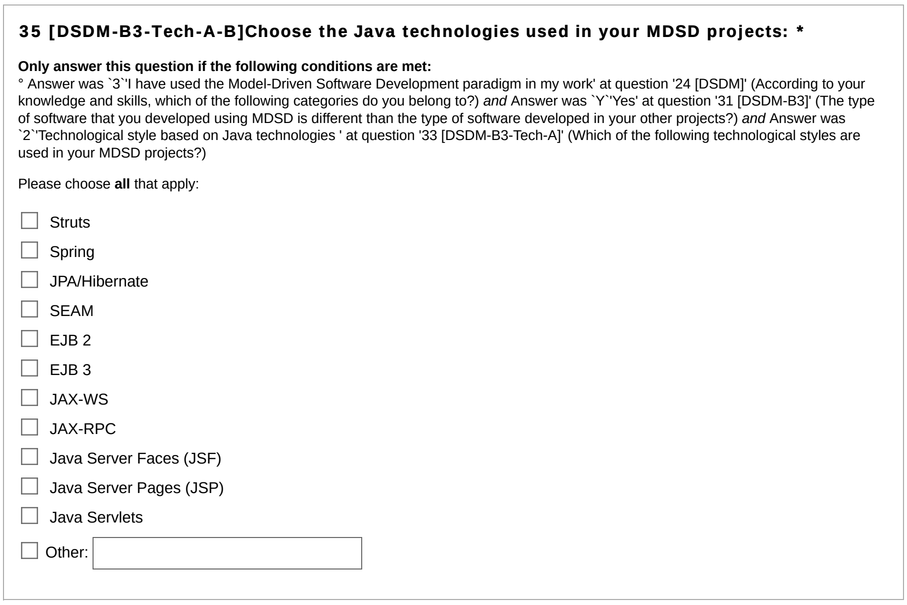
☐ WAMP (Windows, Apache, MySQL, PHP/Perl/Python)

☐ WIMP (Windows, IIS, MySQL, PHP/Perl/Python)

☐ WISA (Windows, IIS, SQL Server, ASP)

☐ OpenACS (Linux/Windows, AOLServer, PostgreSQL/Oracle, Tcl)

☐ Other: ____________



35 [DSDM-B3-Tech-A-B]Choose the Java technologies used in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '31 [DSDM-B3]' (The type of software that you developed using MDSD is different than the type of software developed in your other projects?) *and* Answer was `2`'Technological style based on Java technologies ' at question '33 [DSDM-B3-Tech-A]' (Which of the following technological styles are used in your MDSD projects?)

Please choose **all** that apply:

☐ Struts

☐ Spring

☐ JPA/Hibernate

☐ SEAM

☐ EJB 2

☐ EJB 3

☐ JAX-WS

☐ JAX-RPC

☐ Java Server Faces (JSF)

☐ Java Server Pages (JSP)

☐ Java Servlets

☐ Other: ____________

36 [DSDM-B3-Tech-A-C]Choose the .Net technologies used in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '31 [DSDM-B3]' (The type of software that you developed using MDSD is different than the type of software developed in your other projects?) *and* Answer was `3`'Technological style based on .Net technologies' at question '33 [DSDM-B3-Tech-A]' (Which of the following technological styles are used in your MDSD projects?)

Please choose **all** that apply:

☐ ADO.Net
☐ ASP.Net
☐ WCF - Windows Communication Foundation
☐ WF - Windows Workflow Foundation
☐ WPF - Windows Presentation Foundation
☐ Spring.Net
☐ NHibernate
☐ Windows Forms
☐ Other: ______

37 [DSDM-B4]Do you use a particular type of DBMS in your MDSD projects? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **only one** of the following:

○ Yes
○ No

38 [DSDM-B4-DBMS]Choose the type of data base used in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '37 [DSDM-B4]' (Do you use a particular type of DBMS in your MDSD projects?)

Please choose **all** that apply:

☐ Relational
☐ Multidimensional
☐ Object-Relational
☐ Object-Oriented
☐ Documental
☐ Deductive
☐ XML
☐ Other: ______

39 [DSDM-B4-DBMS-A]Choose the Data Base Management System (DBMS) used in your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '37 [DSDM-B4]' (Do you use a particular type of DBMS in your MDSD projects?)

Please choose **all** that apply:

☐ MySQL
☐ PostgreSQL
☐ Oracle
☐ SQL-Server
☐ DB2
☐ Other: ____________

40 [DSDM-B4-DBMS-B]Choose the relevance of the following DBMS capabilities in your MDSD projects:

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `Y`'Yes' at question '37 [DSDM-B4]' (Do you use a particular type of DBMS in your MDSD projects?)

Please choose the appropriate response for each item:

	None	Marginal	Medium	Important	Critical
Stored procedures	○	○	○	○	○
Triggers	○	○	○	○	○
Schema validation (e.g. checks)	○	○	○	○	○

41 [DSDM-B5]Choose the initiatives used on your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

☐ Model-Driven Architecture (MDA)
☐ Model-Driven Development (MDD/MDSD)
☐ Model-Driven Engineering (MDE)
☐ Other: ____________

42 [DSDM-B6]Choose the platforms that you use on your MDSD projects: *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

☐ Eclipse EMP
☐ AndroMDA
☐ openArchitectureWare
☐ Other:

43 [DSDM-B7]Which Model-Driven Software Development CASE tools you use on your MDSD projects? (Model editors, etc.)

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please write your answer here:

44 [DSDM-Opinion1]Give us your opinion about the following sentences: *

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' or 'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose the appropriate response for each item:

	Worsest case	Worse	Equal	Better	Much better
The quality of the software architecture obtained by a MDSD process in comparison with the quality obtained using traditional methods is...	○	○	○	○	○
The productivity of using a MDSD process in comparison with traditional methods is...	○	○	○	○	○

45 [DSDM-Opinion2]Which characteristics or functionalities do you think that are currently missing on the platforms and tools of Model-Driven Software Development?

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' or 'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please write your answer here:

46 [DSDM-Opinion5-A]Why haven't you applied Model-Driven Software Development in your projects? *

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

☐ I don't belive in models for software development

☐ I don't trust in MDSD

☐ MDSD is not mature enough

☐ MDSD do not fit to the kind of projects I develop

☐ Company policy

☐ Other:

47 [DSDM-Opinion5-A-A]Why MDSD doesn't fit to the kind of projects that you develop?

Only answer this question if the following conditions are met:
° Answer was `2`'I know the concept of Model-Driven Software Development but I don't use it in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?) *and* Answer was `4`'MDSD do not fit to the kind of projects I develop' at question '46 [DSDM-Opinion5-A]' (Why haven't you applied Model-Driven Software Development in your projects?)

Please write your answer here:

48 [DSDM-Opinion5-B]Why did you apply Model-Driven Software Development on your projects? *

Only answer this question if the following conditions are met:
° Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please choose **all** that apply:

- ☐ Company policy
- ☐ At a given moment I started to apply MDSD to all my projects
- ☐ I apply MDSD only to some particular kinds of projects
- ☐ I used MDSD in the past but I finally gave up
- ☐ I'm still experimenting
- ☐ Other: ____________

49 [DSDM-Opinion5-B-A]In which kinds of projects have you applied MDSD?

Only answer this question if the following conditions are met:
° Answer was `3`'I apply MDSD only to some particular kinds of projects' at question '48 [DSDM-Opinion5-B]' (Why did you apply Model-Driven Software Development on your projects?) *and* Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please write your answer here:

50 [DSDM-Opinion5-B-B]Why did you give up MDSD?

Only answer this question if the following conditions are met:
° Answer was `4`'I used MDSD in the past but I finally gave up' at question '48 [DSDM-Opinion5-B]' (Why did you apply Model-Driven Software Development on your projects?) *and* Answer was `3`'I have used the Model-Driven Software Development paradigm in my work' at question '24 [DSDM]' (According to your knowledge and skills, which of the following categories do you belong to?)

Please write your answer here:

Appendix B

Second empirical study

Questionnaire of the second empirical study:

Architectural Practices in relation to NFRs

STUDY OF ARCHITECTURAL PRACTICES IN RELATION TO NON-FUNCTIONAL REQUIREMENTS

The Software Engineering for Information Systems research Group (GESSI) from the Technical University of Catalunya and the Software Engineering Research Group from the Universitat Oberta de Catalunya (UOC), are jointly participating in a research project aimed to investigate the industrial practice of software architecture design. Our ultimate goal is to contribute to fill in the gap among academic research and real industrial practices related to software architecture.

As part of this project, we plan to perform several interviews in IT companies in order to know their practices and needs related to architectural design. Our main interest is to inquiry how architectural design is performed, which roles are involved, and which information and knowledge are required for this labour, as well as to investigate which factors influence design decisions and technologies to be used.

In this context, we are kindly asking for your participation. We need the participation of one or more professionals in charge of software architecture design tasks in any project from your company. The interview will last 1 hour approximately. All provided information will be treated strictly confidential.

The interview will be based on a single project. So, before starting the interview please select any project where you played the role of software architect.

Section 1- About the Respondent

****** To be answered before the interview (if possible) ******

This section contains questions related to the respondent and the respondent's company.

All provided information will be treated ***strictly confidential****.*

Personal information

Name and surname:	
Contact e-mail:	

Related Studies

Principal degree:	
Related studies:	

Professional experience in the company

Position:	
Years in this position:	
Years in the company:	

About the company

Name of the company:	
Number of employees:	
Principal production:	

About the project

Name of the project:	
Domain of the project:	
Number of participants:	
Project duration:	
End date:	
Economic costs:	

Section 2- Questions about the project

****** Questions to be answered during the interview ******

> The projects we want to investigate in this interview are **typical completed software development projects** you were involved in as software architect. If you have experience with several such projects, please **select the project** that you are **most familiar with,** and **base your answers on the system developed in that project**.

Introductory Questions

I1 Could you please provide a general description of the company?

I2 Could you please provide a general description of the project you will base your answers on?

Methodological Aspects

In this subsection we will inquiry about your understanding of the main concepts in relation to software architecture as well as methodological aspects related to the project.

Q1 What do you understand by “software architecture”?

Q2 Do you consider that the role of “software architect” exists in your company? What are the responsibilities of this role?

Q3 Could you please describe the methodology followed for the design of the architecture and the selection of technologies?

Q4 Did you generate documentation related to architectural issues? What kind of documents? Did you use any formalisms, models or languages?

Q5 Did you document any architectural view? Why?

Q6 Did you have limited freedom to make architectural decisions?

Q7 Which type of decisions were you able to make?

Architectural design

This subsection is aimed to inquire about the high-level architectural decisions that you made as well as decisions related to technology. Examples of architectural decisions could be the separation of the functionalities in modules, the type of communication between them, or the selection of the architectural style. Examples of technology selection could be the technology chosen for every component, the communication protocols, the DBMS, or the selection of the technological platform (e.g. .NET).

Q8 Did you follow any order to make architectural and technology decisions? (e.g., First architectural decisions and after the selection of technologies, reverse order, at the same time, in an independent way). Do you consider it as an iterative process?

Q9 Which role played NFR during architectural and technological decision making?

Q10 Which NFRs were considered during architectural and technological decision making?

Q11 Which NFRs had a *major* influence on your decisions? Why?

Q12 Which NFRs had a *minor* influence on your decisions? Why?

In particular, do you consider that there was any type of NFR that was not considered at all? Why?

Q13 Dou you consider that you would made the same decisions if you didn't consider NFRs? Why?

Q14 Which other factors do you consider that played an important role architectural or technological decision making processes? How do they influenced the decisions?

Q15 Do you consider that the NFRs considered in the project were complete?

Did you add some NFRs from your experience?

Did you modify or remove some of the initial NFRs? In any of these cases, did you make the decision on your own? Did you document any changes in the NFRs?

Q16 Did you consider that architectural and technological decision making processes could have been improved and/or make it easier with some tool support? (E.g. modeling tool, simulation tool, etc.)

Q17 Did you consider that the architectural decisions and the selected technologies were able to fulfill the required NFRs?

Appendix C

Third empirical study

Questionnaire of the third empirical study:

The role of quality attributes in service-based systems design

The role of quality attributes in service-based systems design

This survey **studies how practitioners treat quality attributes** (such as performance, security, availability, and other -ilities) when designing service-based software systems. Our **goal** is to understand what decisions practitioners make to accommodate quality attributes and to ensure that service-based systems meet quality goals. **Your participation** will help us understand the real needs when handling quality attributes, and focus our research to improve how quality attributes are treated in real-world systems.
This study is conducted jointly by the Group of Software Engineering for Information Systems (GESSI), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, and the Software Engineering and Architecture Group (SEARCH), University of Groningen (RUG), Netherlands. The investigators are:

- David Ameller	GESSI	UPC	dameller@essi.upc.edu
- Matthias Galster	SEARCH	RUG	m.r.galster@rug.nl
- Paris Avgeriou	SEARCH	RUG	paris@cs.rug.nl
- Xavier Franch	GESSI	UPC	franch@essi.upc.edu

There are 48 questions in this survey

Information about this survey

Please take a minute to read through the information below. It will help you answer the questionnaire.

1 [Note]What will you be asked to do?

This questionnaire consist of **43 questions**, some questions are optional and some questions will only appear depending on your answers to previous questions. The following types of questions will be asked:

- Questions about your profile.
- Questions about one of your previous projects.
- Questions about quality attributes related to architectural design decisions in the selected project.

2 [Note]How long it will take to complete the survey?

The survey should take approximately **20 minutes** to complete.

3 [Note]Dissemination of the results.

If you are interested in the results of this survey, we will provide you with the research report after all data have been collected and analyzed.

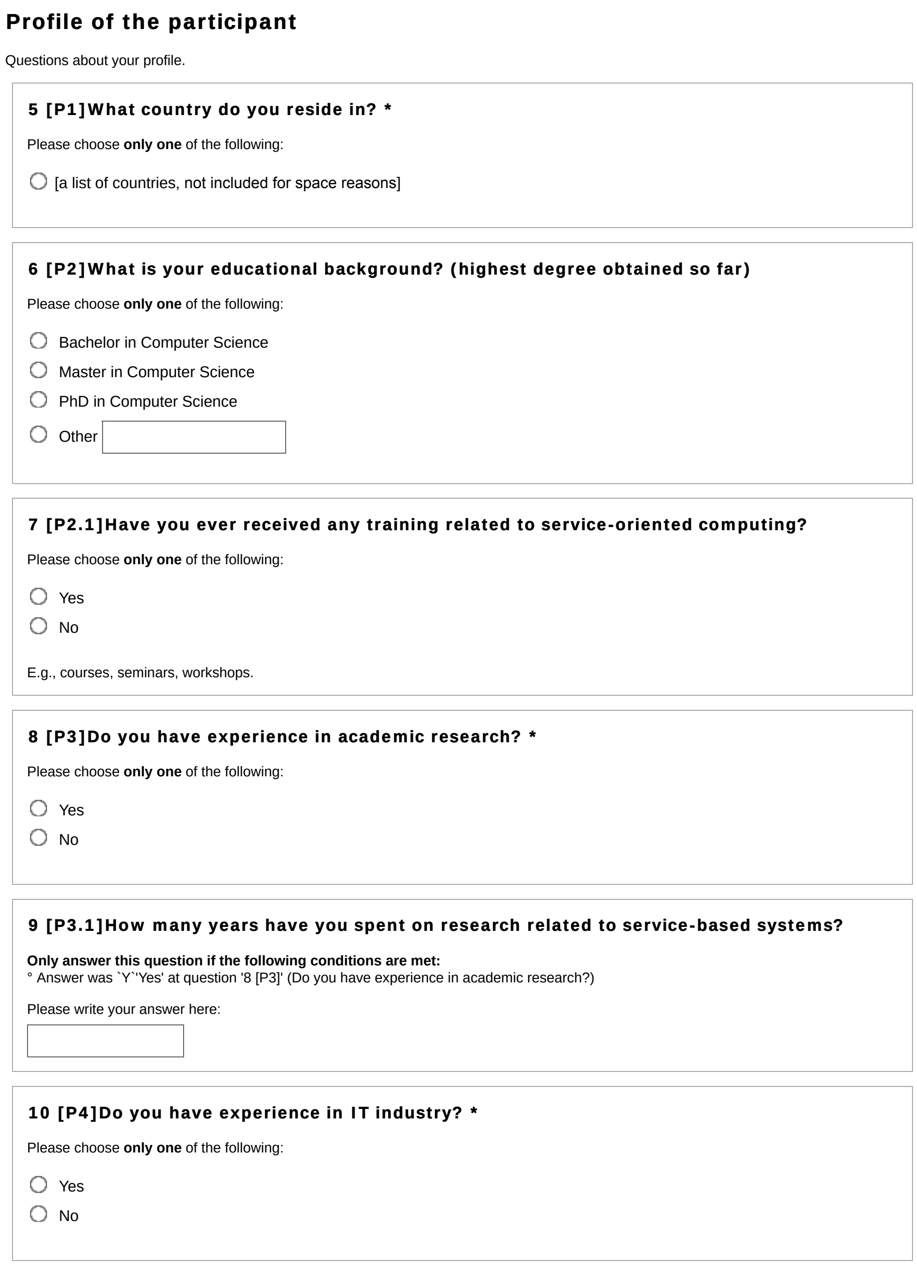
After completing the survey, we will let you know how to obtain the report.

4 [Note]Before you start the survey...

...please take a moment to think about a project in which you were involved in the past.

The project should meet the following criteria:

- It should have utilized service-oriented computing/architecture.
- You took some kind of design/architecture responsibility.



Profile of the participant

Questions about your profile.

5 [P1]What country do you reside in? *

Please choose **only one** of the following:

○ [a list of countries, not included for space reasons]

6 [P2]What is your educational background? (highest degree obtained so far)

Please choose **only one** of the following:

○ Bachelor in Computer Science

○ Master in Computer Science

○ PhD in Computer Science

○ Other

7 [P2.1]Have you ever received any training related to service-oriented computing?

Please choose **only one** of the following:

○ Yes

○ No

E.g., courses, seminars, workshops.

8 [P3]Do you have experience in academic research? *

Please choose **only one** of the following:

○ Yes

○ No

9 [P3.1]How many years have you spent on research related to service-based systems?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '8 [P3]' (Do you have experience in academic research?)

Please write your answer here:

10 [P4]Do you have experience in IT industry? *

Please choose **only one** of the following:

○ Yes

○ No

11 [P4.1]How many years of experience do you have in IT industry? *

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '10 [P4]' (Do you have experience in IT industry?)

Please write your answer here:

12 [P4.2]What is / was your main role in your company? *

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '10 [P4]' (Do you have experience in IT industry?)

Please choose **only one** of the following:

○ Project manager
○ Architect/Designer
○ Developer
○ Other

13 [P4.3]What is / was the size of your company in terms of the number of employees?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '10 [P4]' (Do you have experience in IT industry?)

Please choose **only one** of the following:

○ Less than 10 employees
○ Between 10 and 50 employees
○ Between 50 and 250 employees
○ More than 250 employees

14 [P4.4]What domain is / was your company in?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '10 [P4]' (Do you have experience in IT industry?)

Please choose **all** that apply:

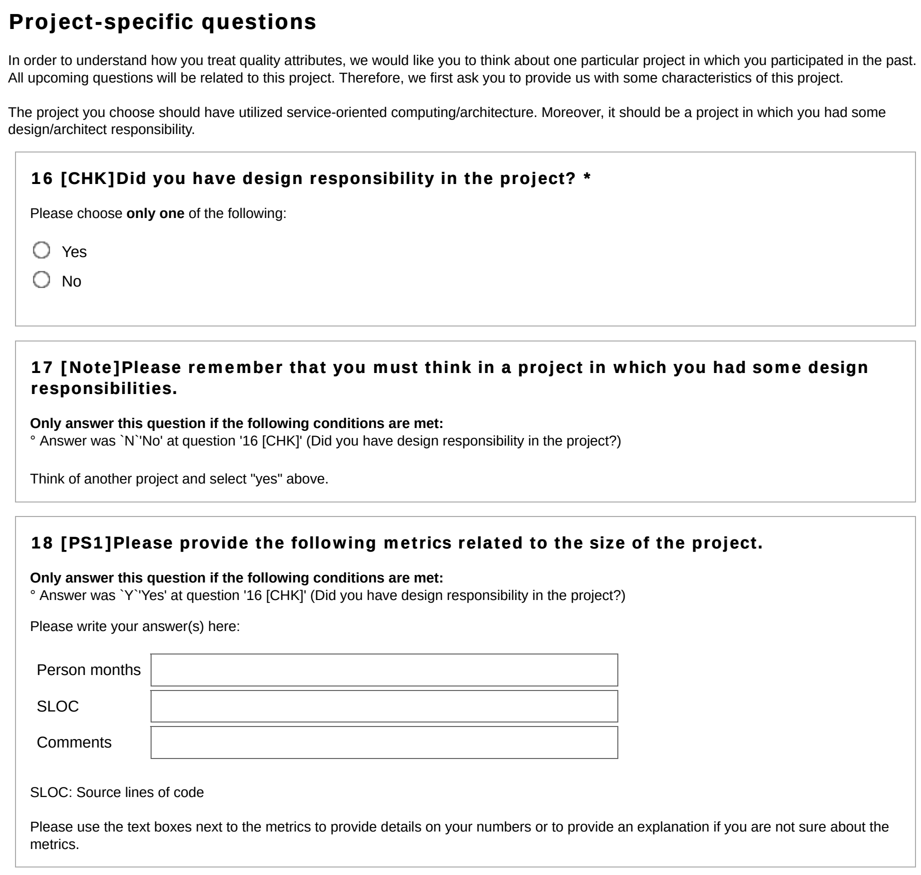
☐ Automotive
☐ Consulting
☐ Customer relationship management
☐ E-commerce
☐ Education
☐ Finance
☐ Government
☐ Healthcare
☐ Human resources
☐ Insurance
☐ Manufacturing
☐ Power distribution
☐ Research and development
☐ Software engineering
☐ Telecommunication
☐ Transportation
☐ Travel
☐ Other: ______

15 [P4.5]How many years have you spent on doing work related to service-oriented computing?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '10 [P4]' (Do you have experience in IT industry?)

Please write your answer here:

______



Project-specific questions

In order to understand how you treat quality attributes, we would like you to think about one particular project in which you participated in the past. All upcoming questions will be related to this project. Therefore, we first ask you to provide us with some characteristics of this project.

The project you choose should have utilized service-oriented computing/architecture. Moreover, it should be a project in which you had some design/architect responsibility.

16 [CHK]Did you have design responsibility in the project? *

Please choose **only one** of the following:

○ Yes
○ No

17 [Note]Please remember that you must think in a project in which you had some design responsibilities.

Only answer this question if the following conditions are met:
° Answer was `N`'No' at question '16 [CHK]' (Did you have design responsibility in the project?)

Think of another project and select "yes" above.

18 [PS1]Please provide the following metrics related to the size of the project.

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please write your answer(s) here:

Person months	
SLOC	
Comments	

SLOC: Source lines of code

Please use the text boxes next to the metrics to provide details on your numbers or to provide an explanation if you are not sure about the metrics.

19 [PS2]What is the domain of the project that you are thinking about? *

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

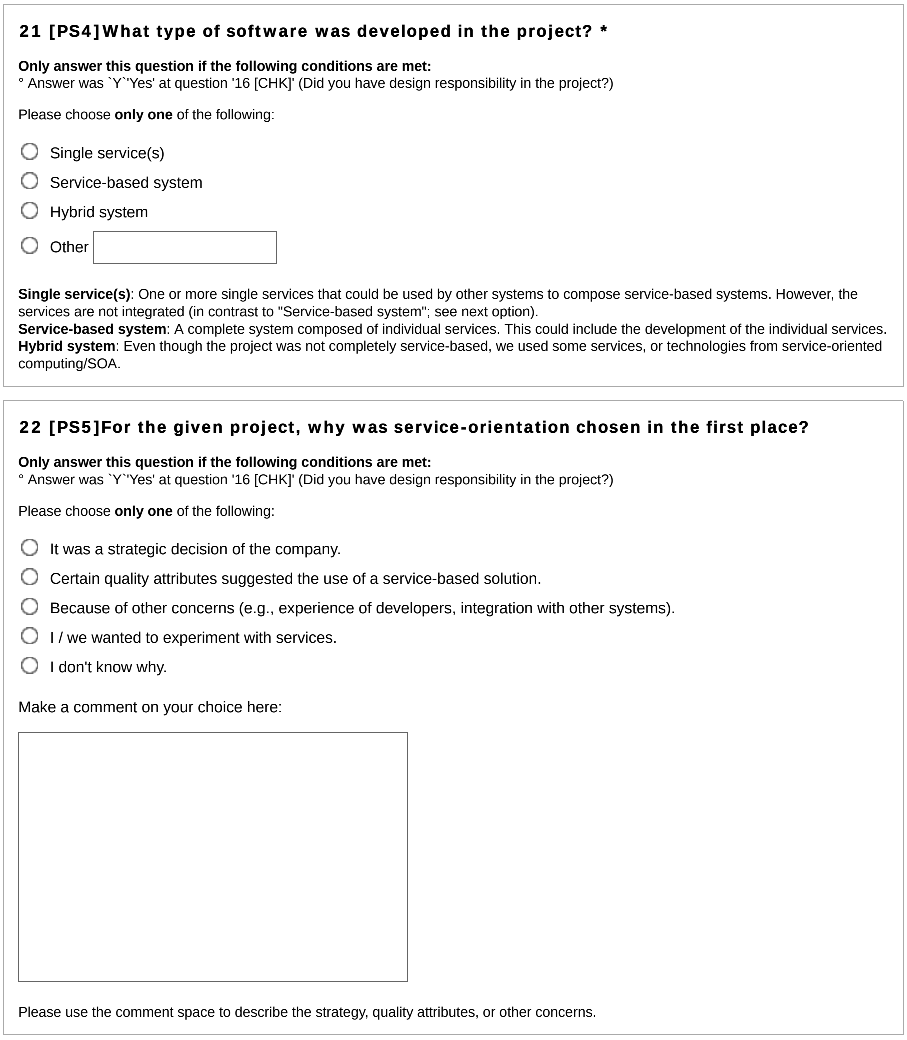
Please choose **only one** of the following:

- ○ Automotive
- ○ Consulting
- ○ Customer relationship management
- ○ E-commerce
- ○ Education
- ○ Embedded systems
- ○ Enterprise computing
- ○ Finance
- ○ Government
- ○ Healthcare
- ○ Human resources
- ○ Insurance
- ○ Manufacturing
- ○ Power distribution
- ○ Research and development
- ○ Software engineering
- ○ Telecommunication
- ○ Transportation
- ○ Travel
- ○ Other

20 [PS3]Please provide a brief description of the project (1 to 2 sentences).

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please write your answer here:



21 [PS4]What type of software was developed in the project? *

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please choose **only one** of the following:

○ Single service(s)
○ Service-based system
○ Hybrid system
○ Other

Single service(s): One or more single services that could be used by other systems to compose service-based systems. However, the services are not integrated (in contrast to "Service-based system"; see next option).
Service-based system: A complete system composed of individual services. This could include the development of the individual services.
Hybrid system: Even though the project was not completely service-based, we used some services, or technologies from service-oriented computing/SOA.

22 [PS5]For the given project, why was service-orientation chosen in the first place?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please choose **only one** of the following:

○ It was a strategic decision of the company.
○ Certain quality attributes suggested the use of a service-based solution.
○ Because of other concerns (e.g., experience of developers, integration with other systems).
○ I / we wanted to experiment with services.
○ I don't know why.

Make a comment on your choice here:

Please use the comment space to describe the strategy, quality attributes, or other concerns.

23 [PS6]Select the sentence that describes the use of external services in your project best.

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please choose **only one** of the following:

- ○ The project did not use external services but only services that were developed in-house.
- ○ The project used external services provided by trusted sources.
- ○ A search for external services was done, not considering any specific source.
- ○ The developed software used a self-adapting mechanism to discover new external services when necessary.

24 [PS7]Compared to functionality, how important were quality attributes when designing the system of the project you are thinking about?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please choose **only one** of the following:

- ○ Quality attributes were not important.
- ○ Quality attributes were less important than functionality.
- ○ Functionality and quality attributes were equally important.
- ○ Quality attributes were more important than functionality.
- ○ Almost all effort was spent to ensure the compliance of quality attributes.
- ○ I don't know.

25 [PS8]Were quality attributes considered implicitly or explicitly?

Only answer this question if the following conditions are met:
° Answer was `Y`'Yes' at question '16 [CHK]' (Did you have design responsibility in the project?)

Please choose **only one** of the following:

- ○ Implicitly (in your project quality attributes existed but you did not consider them as particular requirements)
- ○ Explicitly (quality attributes were made explicit in the requirements)

Quality impact on architectural design decisions

In this part of the survey we ask you about the most important quality attribute and one decision that you made in order to accommodate this quality attribute.

26 [QA1.1]What was the most important quality attribute in your project? *

Please write your answer here:

27 [QA1.2]What part of the system was affected most by this quality attribute? (If necessary provide a brief description of the affected part)

Please write your answer here:

28 [QA1.3]What situations or events had / have to happen to make this quality attribute evident or visible to end users or other stakeholders?

Please write your answer here:

29 [QA1.4]What restrictions or goals were imposed on this quality attribute?

Please write your answer here:

30 [QA1.5]How did you measure or test the satisfaction of this quality attribute (include quantitative information if applicable)?

Please write your answer here:

31 [ADD1.1]What was the most important design decision that you made in the project that is related to this quality attribute? *

Please write your answer here:

Please note that "related" could mean that the design decision was taken in order to accommodate this quality attribute, or that this quality attribute affected or constrained the design decision, or that the quality attribute was significantly affected by this design decision.

32 [ADD1.2]What other alternatives did you consider for this decision?

Please write your answer(s) here:

Alternative 1:	
Alternative 2:	
Alternative 3:	

33 [ADD1.3]What is the reason why you selected this decision? Also, why did you reject the other alternatives?

Please write your answer here:

34 [ADD1.4]Was this decision related or forced by previous decisions? (please comment)

Please write your answer here:

35 [ADD1.5]What other quality attributes were affected (negatively or positively) by this decision, and how?

Please choose **all** that apply:

☐ **Security:** quality attributes related to accountability, traceability/auditability, encryption, non-repudiation, safety, authorization, confidentiality, integrity, and authentication.

☐ **Data-related:** quality attributes related to data validity, data timeliness, data reliability, data completeness, data policy, data integrity, and data accuracy.

☐ **Other:** quality attributes related to the compliance to some standardization effort.

☐ **Configuration and management:** quality attributes related to stability/change cycle, reputation, completeness, and level of service.

☐ **Performance:** quality attributes related to transaction time, throughput, response time, latency, execution time, and queue delay time.

☐ **Quality of use context:** quality attributes related to coverage, up-to-dateness/freshness, trust-worthiness, probability of correctness, precision, and temporal/spatial resolution.

☐ **Usability:** quality attributes related to efficiency of use, content accessibility, learnability, aesthetics and attractiveness, and effectiveness of the operability and navegability.

☐ **Dependability:** quality attributes related to scalability, capacity, reliability, accessibility, availability, failure semantics (e.g., exception handling), accuracy, and robustness/flexibility.

☐ **Cost:** quality attributes related to variable costs, cost model, and fixed costs.

How quality attributes of the system are affected by this decision can be detailed after selecting a quality attribute.

36 [ADD1.5.1]How positive or negative was this decision for the security quality?

Only answer this question if the following conditions are met:
° Answer was `1`'**Security:** quality attributes related to accountability, traceability/auditability, encryption, non-repudiation, safety, authorization, confidentiality, integrity, and authentication.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Accountability	○	○	○	○	○	○
Traceability/auditability	○	○	○	○	○	○
Encryption	○	○	○	○	○	○
Non-repudiation	○	○	○	○	○	○
Safety	○	○	○	○	○	○
Authorization	○	○	○	○	○	○
Confidentiality	○	○	○	○	○	○
Integrity	○	○	○	○	○	○
Authentication	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

37 [ADD1.5.2]How positive or negative was this decision for the data quality?

Only answer this question if the following conditions are met:
° Answer was `2`'**Data-related:** quality attributes related to data validity, data timeliness, data reliability, data completeness, data policy, data integrity, and data accuracy.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Data validity	○	○	○	○	○	○
Data timeliness	○	○	○	○	○	○
Data reliability	○	○	○	○	○	○
Data completeness	○	○	○	○	○	○
Data policy	○	○	○	○	○	○
Data integrity	○	○	○	○	○	○
Data accuracy	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

38 [ADD1.5.3]How positive or negative was this decision for the standards compliance quality?

Only answer this question if the following conditions are met:
° Answer was `3`'**Other:** quality attributes related to the compliance to some standardization effort.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Standards compliance	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

39 [ADD1.5.4]How positive or negative was this decision for the configuration and management quality?

Only answer this question if the following conditions are met:
° Answer was `4`'**Configuration and management:** quality attributes related to stability/change cycle, reputation, completeness, and level of service.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Stability/change cycle	○	○	○	○	○	○
Reputation	○	○	○	○	○	○
Completeness	○	○	○	○	○	○
Level of service	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

40 [ADD1.5.5]How positive or negative was this decision for the performance quality?

Only answer this question if the following conditions are met:
° Answer was `5`'**Performance:** quality attributes related to transaction time, throughput, response time, latency, execution time, and queue delay time.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Transaction time	○	○	○	○	○	○
Throughput	○	○	○	○	○	○
Response time	○	○	○	○	○	○
Latency	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

41 [ADD1.5.6]How positive or negative was this decision for the use context quality?

Only answer this question if the following conditions are met:
° Answer was `6`'**Quality of use context:** quality attributes related to coverage, up-to-dateness/freshness, trust-worthiness, probability of correctness, precision, and temporal/spatial resolution.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Coverage	○	○	○	○	○	○
Up-to-dateness/freshness	○	○	○	○	○	○
Trust-worthiness	○	○	○	○	○	○
Probability of correctness	○	○	○	○	○	○
Precision	○	○	○	○	○	○
Temporal/spatial resolution	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

42 [ADD1.5.7]How positive or negative was this decision for the usability quality?

Only answer this question if the following conditions are met:
° Answer was `7`'**Usability:** quality attributes related to efficiency of use, content accessibility, learnability, aesthetics and attractiveness, and effectiveness of the operability and navegability.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Efficiency of use	○	○	○	○	○	○
Content accessibility	○	○	○	○	○	○
Learnability	○	○	○	○	○	○
Aesthetics and attractiveness	○	○	○	○	○	○
Effectiveness of the operability and navegability	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

43 [ADD1.5.8]How positive or negative was this decision for the dependability quality?

Only answer this question if the following conditions are met:
° Answer was `8`'**Dependability:** quality attributes related to scalability, capacity, reliability, accessibility, availability, failure semantics (e.g., exception handling), accuracy, and robustness/flexibility.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Scalability	○	○	○	○	○	○
Capacity	○	○	○	○	○	○
Reliability	○	○	○	○	○	○
Accessibility	○	○	○	○	○	○
Continuous availability	○	○	○	○	○	○
Availability	○	○	○	○	○	○
Failure semantics	○	○	○	○	○	○
Accuracy	○	○	○	○	○	○
Robustness/flexibility	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

44 [ADD1.5.9]How positive or negative was this decision for the cost quality?

Only answer this question if the following conditions are met:
° Answer was `9`'**Cost:** quality attributes related to variable costs, cost model, and fixed costs.' at question '35 [ADD1.5]' (What other quality attributes were affected (negatively or positively) by this decision, and how?)

Please choose the appropriate response for each item:

	Very negative	Negative	No effect	Positive	Very positive	Not applicable
Variable costs	○	○	○	○	○	○
Cost model	○	○	○	○	○	○
Fixed costs	○	○	○	○	○	○

If you have doubts of the meaning of any of these quality attributes place the mouse pointer over its name to get a definition.

45 [ADD1.5.10]You can use this comment space to make any comment on the quality questions or to mention quality attributes that are not listed.

Please write your answer here:

Closing questions

These are the last questions of the survey. They give you an opportunity to provide us with further information that was not covered by the previous questions.

46 [C1]In your projects, do you usually document information about design decisions? Also, do you think it is or it would be useful to have this information? (Please comment)

Please write your answer here:

47 [C2]What problems do you think occur when you try to satisfy quality attributes in the context of service-based systems (if any)?

Please write your answer here:

48 [C3]Upon reflection of answering the questions, is there anything you can add and that you feel is relevant in the context of this questionnaire?

Please write your answer here:

List of Abbreviations

AD	Architectural Decision
ADD	Architectural Design Decision
ADL	Architectural Description Language
AK	Architectural Knowledge
AS	Application Server
CIM	Computation Independent Model
CRUD	Create, Read, Update and Delete
DBMS	Data Base Management System
GORE	Goal-Oriented Requirements Engineering
KB	Knowledge Base
M2M	Model-to-Model
M2T	Model-to-Text
MDA	Model-Driven Architecture
MDD	Model-Driven Development

MDE	Model-Driven Engineering
NFR	Non-Functional Requirement
OSS	Open Source Software
PIM	Platform Independent Model
PSM	Platform Specific Model
QA	Quality Attribute
QR	Quality Requirement
RE	Requirements Engineering
RQ	Research Question
SADM	Software Architectural Design Method
SBS	Service-Based System
SEI	Software Engineering Institute
SLA	Service-Level Agreement
SOA	Service-Oriented Architecture
UML	Unified Modeling Language
WS	Web Server

Bibliography

[1] Art Akerman and Jeff Tyree. Using ontology to support development of software architectures. *IBM Systems Journal*, 45:813–825, October 2006.

[2] Tariq Al-naeem, Ian Gorton, Muhammed Ali-Babar, Fethi Rabhi, and Boualem Benatallah. A quality-driven systematic approach for architecting distributed software applications. In *27th International Conference on Software Engineering (ICSE)*, pages 244–253, 2005.

[3] Fernanda Alencar, Beatriz Marín, Giovanni Giachetti, Oscar Pastor, Jaelson Castro, and João Henrique Pimentel. From *i** Requirements Models to Conceptual Models of a Model Driven Development Process. In *2nd Working Conference on the Practice of Enterprise Modeling (PoEM)*, pages 99–114, 2009.

[4] Muhammad Ali-Babar and Ian Gorton. A Tool for Managing Software Architecture Knowledge. In *2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*, page 11, 2007.

[5] Muhammad Ali-Babar, Ian Gorton, and Barbara Kitchenham. *Rationale Management in Software Engineering*, chapter A Framework for Supporting Architecture Knowledge and Rationale Management, pages 237–254. Springer, 2006.

[6] Muhammad Ali-Babar, Patricia Lago, and Arie Deursen. Empirical research in software architecture: opportunities, challenges, and approaches. *Empirical Software Engineering*, 16:539–543, 2011.

[7] Muhammad Ali-Babar, Andrew Northway, Ian Gorton, Paul Heuer, and Thong Nguyen. Introducing Tool Support for Managing Architectural Knowledge: An Experience Report. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 105–113, 2008.

[8] David Ameller. BSc thesis: Assignació de responsabilitats usant AndroMDA. Technical report, Facultat d'Informàtica de Barcelona (FIB), January 2007. http://upcommons.upc.edu/pfc/handle/2099.1/5302.

[9] David Ameller. MSc thesis: Considering Non-Functional Requirements in Model-Driven Engineering. Technical report, Llenguatges i Sistemes Informátics (LSI), June 2009. http://upcommons.upc.edu/pfc/handle/2099.1/7192.

[10] David Ameller. PhD thesis proposal: Systematic Architecture Design, a semi-automatic method. Technical report, Llenguatges i Sistemes Informátics (LSI), June 2010. http://www.essi.upc.edu/ dameller/wp-content/papercite-data/pdf/ameller2010-proposal.pdf.

[11] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do Software Architects Consider Non-functional Requirements: An Exploratory Study. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 41–50. IEEE, 2012.

[12] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. Non-functional Requirements in Architectural Decision Making. *IEEE Software*, 30(2):61–67, March 2013. Date of Publication: 13 December 2012.

[13] David Ameller, Oriol Collell, and Xavier Franch. Reconciling the 3-layer Architectural Style with a Plug-in-based Architecture: the Eclipse Case. In *1st Workshop on Developing Tools as Plug-ins (TOPI)*, pages 20–23, 2011.

[14] David Ameller, Oriol Collell, and Xavier Franch. ArchiTech: Tool Support for NFR-Guided Architectural Decision-Making. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 315–316. IEEE, 2012.

[15] David Ameller, Oriol Collell, and Xavier Franch. The Three-Layer architectural pattern applied to plug-in-based architectures: the Eclipse case. *Software: Practice and Experience*, 43:391–402, 2012.

[16] David Ameller and Xavier Franch. Asignación de Tratamientos a Responsabilidades en el contexto del Diseño Arquitectónico Dirigido por Modelos. In *Workshop on Desarrollo de Software Dirigido por Modelos (DSDM)*, 2007.

[17] David Ameller and Xavier Franch. Assigning Treatments to Responsibilities in Software Architectures. In *EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Research in progess track*, 2007.

[18] David Ameller and Xavier Franch. Service Level Agreement Monitor (SALMon). In *7th International Conference on Composition-Based Software Systems (ICCBSS)*, pages 224–227, 2008.

[19] David Ameller and Xavier Franch. Definición de una Ontología para el Proceso de DSDM considerando Requisitos No-Funcionales. In *Workshop on Desarrollo de Software Dirigido por Modelos (DSDM)*, 2009.

[20] David Ameller and Xavier Franch. Usage of architectural styles and technologies in IT companies and organizations. In *Workshop on Empirical Assessment in Software Architecture (EASA)*, 2009.

[21] David Ameller and Xavier Franch. How do Software Architects consider Non-Functional Requirements: A Survey. In *16th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ)*, pages 276–277, 2010.

[22] David Ameller and Xavier Franch. Ontology-based Architectural Knowledge representation: structural elements module. In *9th*

International Workshop on System/Software Architectures (IWSSA), pages 296–301, 2011.

[23] David Ameller and Xavier Franch. Quark: a method to assist software architects in architectural decision-making. In *XVI Congreso Iberoamericano en Ingeniería de Software (CIbSE)*, 2013.

[24] David Ameller, Xavier Franch, and Jordi Cabot. Dealing with Non-Functional Requirements in Model-Driven Development. In *18th IEEE International Requirements Engineering Conference (RE)*, pages 189–198, 2010.

[25] David Ameller, Matthias Galster, Paris Avgeriou, and Xavier Franch. The Role of Quality Attributes in Service-based Systems Architecting: A Survey. In *7th European Conference on Software Architecture*, pages 200–207, 2013.

[26] Nguyen Duc Anh, Daniela S. Cruzes, Reidar Conradi, Martin Höst, Xavier Franch, and Claudia P. Ayala. Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components. In *18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 77–93, 2012.

[27] Danilo Ardagna, Carlo Ghezzi, and Raffaela Mirandola. Rethinking the Use of Models in Software Architecture. In S. Becker and F. Plasil, editors, *4th International Conference on the Quality of Software Architectures (QoSA)*, volume LNCS 5281, pages 1–27. Springer-Verlag Berlin, 2008.

[28] Colin Atkinson and Thomas Kühne. Model-Driven Development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, September 2003.

[29] Paris Avgeriou, Philippe Kruchten, Patricia Lago, Paul Grisham, and Dewayne Perry. Architectural knowledge and rationale: issues, trends, challenges. *ACM SIGSOFT Software Engineering Notes*, 32:41–46, July 2007.

[30] Paris Avgeriou and Uwe Zdun. Architectural Patterns Revisited - a Pattern Language. In *10th European Conference on Pattern Languages of Programs (EuroPLoP 2005)*, pages 1–39, Irsee, Germany, July 2005.

[31] Lenin Babu T., M. Seetha Ramaiah, T. V. Prabhakar, and D. Rambabu. ArchVoc - Towards an Ontology for Software Architecture. In *2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*, SHARK-ADI '07, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[32] Felix Bachmann and Len Bass. Introduction to the Attribute Driven Design method. In *23rd International Conference on Software Engineering (ICSE)*, pages 745–746, Washington, DC, USA, 2001. IEEE Computer Society.

[33] Felix Bachmann, Len Bass, M. Klein, and C. Shelton. Designing software architectures to achieve quality attribute requirements. *IEEE Software*, 152(4):153–165, August 2005. Article.

[34] Sriram Balasubramaniam, Grace A. Lewis, Edwin Morris, Soumya Simanta, and Dennis B. Smith. Challenges for assuring quality of service in a service-oriented environment. In *Principles of Engineering Service Oriented Systems (PESOS)*, pages 103–106, 2009.

[35] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[36] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[37] Lúcia Bastos and Jaelson Castro. Systematic Integration Between Requirements and Architecture. In Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems III*, volume 3390 of *Lecture*

Notes in Computer Science, pages 85–103. Springer Berlin / Heidelberg, 2005.

[38] Hanane Becha and Daniel Amyot. Non-Functional Properties in Service Oriented Architecture - A Consumer's Perspective. *Journal of Software*, 7(3):575–587, March 2012.

[39] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML Class Diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

[40] Simona Bernardi, José Merseguer, and Dorina C. Petriu. Adding Dependability Analysis Capabilities to the MARTE Profile. In *11th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 736–750, 2008.

[41] Richard Berntsson-Svensson, Tony Gorschek, and Björn Regnell. Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems. In *15th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 218–232, 2009.

[42] Richard Berntsson-Svensson, Tony Gorschek, Björn Regnell, Richard Torkar, Ali Shahrokni, Robert Feldt, and Aybüke Aurum. Prioritization of quality requirements: State of practice in eleven companies. In *19th IEEE International Requirements Engineering Conference (RE)*, pages 69–78, 2011.

[43] Richard Berntsson-Svensson, M. Höst, and B. Regnell. Managing Quality Requirements: A Systematic Review. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 261–268, 2010.

[44] Antonia Bertolino, Antonio Bucchiarone, Stefania Gnesi, and Henry Muccini. An architecture-centric approach for producing quality systems. In R. Reussner, J. Mayer, JA Stafford, S. Overhage, S. Becker, and PJ Schroeder, editors, *Quality Of Software Architectures And Software Quality (QoSA)*, volume 3712, pages 21–37, 2005.

[45] Stephan Bode, Anja Fischer, Winfried Kuehnhauser, and Matthias Riebisch. Software Architectural Design meets Security Engineering. In *16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 109–118, 2009.

[46] Andreas Borg, Angela Yong, Pär Carlshamre, and Kristian Sandahl. The Bad Conscience of Requirements Engineering: An Investigation in Real-world Treatment of Non-Functional Requirements. In *3rd Conference on Software Engineering Research and Practice in Sweden (SERPS)*, pages 1–8, 2003.

[47] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[48] Frederick P. Brooks, Jr. No Silver Bullet - Essence and Accidents of Software Engineering. *Computer*, 20(4):10–19, April 1987.

[49] Frank Buschmann. Value-Focused System Quality. *IEEE Software*, 27(6):84–86, 2010.

[50] Frank Buschmann, David Ameller, Claudia P. Ayala, Jordi Cabot, and Xavier Franch. Architecture Quality Revisited. *IEEE Software*, 29:22–24, 2012.

[51] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns.* John Wiley & Sons, Inc., 1996.

[52] Jordi Cabot and Eric Yu. Improving requirements specifications in Model-Driven Development processes. In *International Workshop on Challenges in Model-Driven Software Engineering (ChaMDE)*, 2008.

[53] Coral Calero, Julián Ruiz, and Mario Piattini. Classifying web metrics using the web quality model. *Online Information Review*, 29(3):227–248, 2005.

[54] Rafael Capilla, Francisco Nava, and Juan C. Dueñas. Modeling and Documenting the Evolution of Architectural Design Decisions. In *2nd Workshop on SHAring and Reusing architectural Knowledge*

Architecture, Rationale, and Design Intent (SHARK-ADI), page 9, Washington, DC, USA, 2007. IEEE Computer Society.

[55] Rafael Capilla, Francisco Nava, Sandra Pérez, and Juan C. Dueñas. A web-based tool for managing architectural design decisions. *ACM SIGSOFT Software Engineering Notes*, 31(5):4, 2006.

[56] Juan-Pablo Carvallo, Xavier Franch, and Carme Quer. Managing non-technical requirements in COTS components selection. In *14th IEEE International Conference on Requirements Engineering*, pages 323–326. IEEE, 2006.

[57] Jaelson Castro, João Pimentel, Márcia Lucena, Emanuel Santos, and Diego Dermeval. F-STREAM: A Flexible Process for Deriving Architectures from Requirements Models. In *International Workshop on System/Software Architectures (IWSSA)*, pages 342–353, 2011.

[58] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[59] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems. In Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[60] Heeseok Choi, Keunhyuk Yeom, Youhee Choi, and Mikyeong Moon. An approach to quality achievement at the architectural level: AQUA. In R. Gorrieri and H. Wehrheim, editors, *8th IFIP*

International Conference on Formal Methods for Open Object-Based Distributed Systems, volume 4037, pages 20–32, 2006.

[61] Lawrence Chung, Kendra Cooper, and Anna Yi. Developing adaptable software architectures using design patterns: an NFR approach. *Computer Standards & Interfaces*, 25(3):253–260, 2003.

[62] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. *Conceptual Modeling: Foundations and Applications*, chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer, 2009.

[63] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering.* Kluwer Academic, 2000.

[64] M. Dal Cin. Extending UML towards a useful OO-language for modeling dependability features. In *9th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 325–330, 2003.

[65] Marcus Ciolkowski, Oliver Laitenberger, Sira Vegas, and Stefan Biffl. Practical Experiences in the Design and Conduct of Surveys in Empirical Software Engineering. In Reidar Conradi and AlfInge Wang, editors, *Empirical Methods and Studies in Software Engineering*, volume 2765 of *Lecture Notes in Computer Science*, pages 104–128. Springer Berlin Heidelberg, 2003.

[66] Paul Clements. Certified Software Architects. *IEEE Software*, 27(6):6–8, 2010.

[67] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating software architectures: methods and case studies.* Addison-Wesley Longman Publishing Co., Inc, 2002.

[68] Viktor Clerc, Patricia Lago, and Hans van Vliet. The architect's mindset. In *3rd International Conference on the Quality of Software-Architectures (QoSA)*, QoSA'07, pages 231–249, Berlin, Heidelberg, 2007.

[69] Sholom Cohen and Robert Krut. Managing Variation in Services in a Software Product Line Context. Technical report, Carnegie Mellon University, 2010.

[70] Reiclar Conradi, Jingyue Li, Odd Petter Petter N Slyngstad, Vigdis By Kampenes, Christian Bunse, Maurizio Morisio, and Marco Torchiano. Reflections on conducting an international survey of software engineering. In *International Symposium on Empirical Software Engineering*, page 10, 2005.

[71] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. Integrating Performance and Reliability Analysis in a Non-Functional MDA Framework. In *Fundamental Approaches to Software Engineering (FASE)*, pages 57–71, 2007.

[72] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. Non-Functional Modeling and Validation in Model-Driven Architecture. In *6th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, page 25, Washington, DC, USA, 2007.

[73] J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (3rd Ed.).* SAGE Publications, 2008.

[74] Maya Daneva, Luigi Buglione, and Andrea Herrmann. Software Architects' Experiences of Quality Requirements: What We Know and What We Do Not Know? In *19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7830 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013.

[75] Alan M. Davis. *Software requirements: objects, functions, and states.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[76] Remco De Boer, Rik Farenhorst, Patricia Lago, Hans van Vliet, Viktor Clerc, and Anton Jansen. Architectural Knowledge: Getting to the Core. In Sven Overhage, Clemens Szyperski, Ralf Reussner, and Judith Stafford, editors, *Software Architectures, Components, and Applications*, volume 4880 of *Lecture Notes in Computer Science*, pages 197–214. Springer Berlin / Heidelberg, 2007.

[77] Remco C. De Boer and Rik Farenhorst. In search of 'architectural knowledge'. In *3rd international workshop on Sharing and reusing architectural knowledge (SHARK)*, pages 71–78, New York, NY, USA, 2008.

[78] Remco C. De Boer, Patricia Lago, Alexandru C. Telea, and Hans Van Vliet. Ontology-driven visualization of architectural design decisions. In *Joint Working IEEE/IFIP Conference on Software Architecture, European Conference on Software Architecture (WICSA/ECSA)*, pages 51–60, sept. 2009.

[79] José Luis de la Vara, Krzysztof Wnuk, Richard Berntsson-Svensson, Juan Sánchez, and Björn Regnell. An Empirical Study on the Importance of Quality Requirements in Industry. In *23th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2011.

[80] Diego Dermeval, Jaelson Castro, Carla T. L. L. Silva, João Pimentel, Ig Ibert Bittencourt, Patrick Henrique da S. Brito, Endhe Elias, Thyago Tenório, and Alan Pedro da Silva. On the use of metamodeling for relating requirements and architectural design decisions. In *ACM Symposium on Applied Computing (SAC)*, pages 1278–1283, 2013.

[81] Diego Dermeval, João Pimentel, Carla T. L. L. Silva, Jaelson Castro, Emanuel Santos, Gabriela Guedes, Márcia Lucena, and Anthony Finkelstein. STREAM-ADD - Supporting the Documentation of Architectural Design Decisions in an Architecture Derivation Process. In *36th Annual IEEE Computer Software and Applications Conference (COMPSAC)*, pages 602–611, 2012.

[82] Jens Dietrich and Chris Elgar. Towards a web of patterns. *Journal of Web Semantics*, 5(2):108–116, 2007.

[83] Joerg Doerr, Daniel Kerkow, Tom Koenig, Thomas Olsson, and Takeshi Suzuki. Non-Functional Requirements in Industry - Three Case Studies Adopting an Experience-based NFR Method. In *13th IEEE Requirements Engineering Conference (RE)*, pages 373–382, 2005.

[84] Thomas Erl. *Service-oriented architecture: concepts, technology, and design.* Prentice Hall, 2005.

[85] Joerg Evermann and Jennifer Fang. Evaluating ontologies: Towards a cognitive measure of quality. *Information Systems*, 35(4):391–403, 2010.

[86] Davide Falessi, Giovanni Cantone, and Philippe Kruchten. Do Architecture Design Methods Meet Architects' Needs? In *6th Working IEEE/IFIP Conference on Software Architecture*, page 5, 2007.

[87] Kazi Farooqui, Luigi Logrippo, and Jan de Meer. The ISO Reference Model for Open Distributed Processing: An Introduction. *Computer Networks and ISDN Systems*, 27(8):1215–1229, 1995.

[88] Agung Fatwanto and Clive Boughton. Analysis, Specification and Modeling of Non-Functional Requirements for Translative Model-Driven Development. In *International Conference on Computational Intelligence and Security*, volume 1 and 2, pages 966–971. IEEE Computer, 2008.

[89] Adrian Fernandez, Emilio Insfran, and Silvia Abrahão. Integrating a Usability Model into Model-Driven Web Development Processes. In *Web Information Systems Engineering (WISE)*, pages 497–510, 2009.

[90] Anthony Finkelstein and John Dowell. A comedy of errors: the London Ambulance Service case study. In *8th International Workshop on Software Specification and Design*, IWSSD '96, page 2, Washington, DC, USA, 1996. IEEE Computer Society.

[91] Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[92] Xavier Franch, Angelo Susi, Maria C. Annosi, Claudia Ayala, Ruediger Glott, Daniel Gross, Ron Kenett, Fabio Mancinelli, Pop Ramsamy, Cedric Thomas, David Ameller, Stijn Bannier, Nili Bergida, Yehuda Blumenfeld, Olivier Bouzereau, Dolors Costal,

Manuel Domínguez, Kirsten Haaland, Lidia López, Mirko Morandini, and Alberto Siena. Managing Risk in Open Source Software Adoption. In *International Joint Conference on Software Technologies (ICSOFT)*, 2013.

[93] Stefano Gallotti, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Quality Prediction of Service Compositions through Probabilistic Model Checking. In S. Becker and F. Plasil, editors, *4th International Conference on the Quality of Software Architectures (QoSA)*, volume LNCS 5281, pages 119–134. Springer-Verlag Berlin, 2008.

[94] David Garlan, Robert Monroe, and David Wile. ACME: an architecture description interchange language. In *Conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, page 7, 1997.

[95] David Garlan and Bradley Schmerl. AEvol: A tool for defining and planning architecture evolution. In *31st International Conference on Software Engineering (ICSE)*, pages 591–594, Washington, DC, USA, 2009.

[96] Dragan Gasevic, Dragan Djuric, Vladan Devedzic, and Violeta Damjanovi. Converting UML to OWL ontologies. In *13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 488–489, New York, NY, USA, 2004. ACM.

[97] Andreas Gehlert and Andreas Metzger. Quality Reference Model for SBA (Deliverable CD-JRA-1.3.2). Technical report, S-Cube, 2009.

[98] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference (RE)*, pages 21–26, 2007.

[99] Laszlo Gonczy, Zsolt Deri, and Daniel Varro. Model Transformations for Performability Analysis of Service Configurations. In MRV Chaudron, editor, *Models in Software Engineering (MiSE)*, volume LNCS 5421, pages 153–166. Springer-Verlag Berlin, 2009.

[100] Ian Gorton. *Essential Software Architecture (2nd ed.)*. Springer, 2011.

[101] Daniel Gross and Eric Yu. From Non-Functional Requirements to Design Through Patterns. *Requirements Engineering Journal*, 6(1):18–36, 2001.

[102] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, December 1995.

[103] Paul Grünbacher, Alexander Egyed, and Nenad Medvidovic. Reconciling software requirements and architectures with intermediate models. *Software and Systems Modeling*, 3(3):235–253, 2004.

[104] Qing Gu and Hans van Vliet. SOA decision making - what do we need to know. In *Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, SHARK '09, pages 25–32, Washington, DC, USA, 2009. IEEE Computer Society.

[105] Nicola Guarino. Some Ontological Principles for Designing Upper Level Lexical Resources. In *1st International Conference on Language Resources and Evaluation (LREC)*, volume cmp-lg/9809002, pages 527–534, 1998.

[106] Giancarlo Guizzardi, Gerd Wagner, and Heinrich Herre. On the Foundations of UML as an Ontology Representation Language. In *14th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 47–62. Springer-Verlag, 2004.

[107] Maria Haigh. Software quality, non-functional software requirements and IT-business alignment. *Software Quality Control*, 18(3):361–385, 2010.

[108] Brent Hailpern and Peri Tarr. Model-Driven Development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–461, 2006.

[109] Katja Henttonen and Mari Matinlassi. Open source based tools for sharing and reuse of software architectural knowledge. In *8th Working IEEE/IFIP Conference on Software Architecture (WICSA) and the 3rd European Conference on Software Architecture (ECSA)*, pages 41–50, 2009.

[110] Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, and Pierre America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, 2007.

[111] Johan F. Hoorn, Rik Farenhorst, Patricia Lago, and Hans van Vliet. The lonesome architect. *Journal of Systems and Software*, 84(9):1424–1435, 2011.

[112] Jennifer Horkoff and Eric S. K. Yu. Qualitative, Interactive, Backward Analysis of *i** Models. In *3rd International i* Workshop*, pages 43–46, 2008.

[113] Leo Hsu and Regina Obe. Cross Compare of SQL Server, MySQL, and PostgreSQL `http://www.postgresonline.com/journal/archives/51-Cross-Compare-of-SQL-Server,-MySQL,-and-PostgreSQL.html`, 2008.

[114] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *33rd International Conference on Software Engineering*, ICSE '11, pages 633–642, New York, NY, USA, 2011. ACM.

[115] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[116] ISO/IEC 25000. Software product Quality Requirements and Evaluation (SQuaRE), 2005.

[117] ISO/IEC 9126. Product quality – Part 1: Quality model, 2001.

[118] ISO/IEC/(IEEE) 42010. 1471-2000: Systems and Software engineering – Recomended practice for architectural description of software-intensive systems, 2007.

[119] Michael Jackson. *Problem frames: analyzing and structuring software development problems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[120] Ivar Jacobson, Grady Booch, and James E. Rumbaugh. *The unified software development process - the complete guide to the unified process from the original designers.* Addison-Wesley object technology series. Addison-Wesley Longman Publishing Co., Inc., 1999.

[121] Anton Jansen and Jan Bosch. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, WICSA '05, pages 109–120, Washington, DC, USA, 2005.

[122] Anton Jansen, Tjaard Vries, Paris Avgeriou, and Martijn Veelen. Sharing the Architectural Knowledge of Quantitative Analysis. In *4th International Conference on Quality of Software-Architectures (QoSA)*, pages 220–234, Berlin, Heidelberg, 2008.

[123] Ivan J Jureta, John Mylopoulos, and Stéphane Faulkner. A core ontology for requirements. *Applied Ontology*, 4(3):169–244, 2009.

[124] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *5th International Conference on The Unified Modeling Language (UML)*, pages 412–425, London, UK, 2002.

[125] Jan Jürjens. *Secure Systems Development with UML.* Springer, 2004.

[126] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements. Scenario-Based Analysis of Software Architecture. *IEEE Software*, 13(6):47–55, 1996.

[127] Rick Kazman and Len Bass. Toward Deriving Software Architectures from Quality Attributes. Technical Report CMU/SEI-94-TR-10, Carnegie Mellon University, 1994.

[128] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems (ICECCS)*, page 68, 10-14 1998.

[129] Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, and Sooyong Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8, Sp. Iss. SI):1211–1231, 2009.

[130] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[131] Barbara Kitchenham. Procedures for Performing Systematic Reviews. Technical report, Keele University, 2004.

[132] Barbara Kitchenham and Shari Lawrence Pfleeger. Principles of survey research - part 6: data analysis. *ACM SIGSOFT Software Engineering Notes*, 28(2):24–27, March 2003.

[133] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research - part 2: designing a survey. *ACM SIGSOFT Software Engineering Notes*, 27(1):18–20, January 2002.

[134] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research - part 3: constructing a survey instrument. *ACM SIGSOFT Software Engineering Notes*, 27(2):20–24, March 2002.

[135] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, August 2002.

[136] Sascha Konrad, Heather J. Goldsby, and Betty H. C. Cheng. i2MAP: An Incremental and Iterative Modeling and Analysis Process. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 451–466, 2007.

[137] Artemios Kontogogos and Paris Avgeriou. An Overview of Software Engineering Approaches to Service Oriented Architectures in Various

Fields. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, pages 254–259, 2009.

[138] Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology.* SAGE Publications, 2004.

[139] Klaus Krogmann and Steffen Becker. A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor. *Software Engineering*, 106:169–176, 2007.

[140] Per Kroll and Philippe Kruchten. *The rational unified process made easy: a practitioner's guide to the RUP.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[141] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[142] Philippe Kruchten. The Software Architect. In *1st Working IFIP Conference on Software Architecture (WICSA)*, pages 565–584, 1999.

[143] Philippe Kruchten. An Ontology of Architectural Design Decisions in Software Intensive Systems. In *2nd Groningen Workshop Software Variability*, pages 54–61, October 2004.

[144] Philippe Kruchten, Rafael Capilla, and Juan Carlos Duenas. The Decision View's Role in Software Architecture Practice. *IEEE Software*, 26:36–42, 2009.

[145] Philippe Kruchten, Patricia Lago, and Hans van Vliet. Building Up and Reasoning About Architectural Knowledge. In *2nd International Conference on Quality of Software Architectures (QoSA)*, pages 43–58, 2006.

[146] Stefan Kugele, Wolfgang Haberl, Michael Tautschnig, and Martin Wechs. Optimizing Automatic Deployment Using Non-functional Requirement Annotations. In T. Margaria and B. Steffen, editors, *Levraging Applications of Formal Methods, Verification and Validation*, volume 17, pages 400–414. Springer-Verlag Berlin, OCT 13-15, 2008 2008.

[147] Claudia Lopez, Luiz Marcio Cysneiros, and Hernan Astudillo. NDR Ontology: Sharing and Reusing NFR and Design Rationale Knowledge. In *1st International Workshop On Managing Requirements Knowledge (MARK)*, pages 1–10, 2008.

[148] Márcia Lucena, Jaelson Castro, Carla T. L. L. Silva, Fernanda M. R. Alencar, and Emanuel Santos. STREAM: A Strategy for Transition between Requirements models and Architectural Models. In *ACM Symposium on Applied Computing (SAC)*, pages 699–704, 2011.

[149] Sara Mahdavi-Hezavehi, Matthias Galster, and Paris Avgeriou. Variability in quality attributes of service-based software systems: A systematic literature review. *Information and Software Technology*, 55(2):320–343, 2013.

[150] Silverio Martínez-Fernández, Claudia Ayala, Xavier Franch, Helena Martins Marques, and David Ameller. Framework for Software Reference Architecture Analysis and Review. In *10th Experimental Software Engineering Track Workshop (ESELAW)*, 2013.

[151] Mari Matinlassi, Eila Niemelä, and Liliana Dobrica. Quality-driven architecture design and analysis method. A revolutionary initiation approach to a product line architecture. In *VTT Technical Research Centre of Finland, VTT Publications 456*, page 128, 2002.

[152] Jose-Norberto Mazón, Jesús Pardillo, and Juan Trujillo. A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. In *15th IEEE International Requirements Engineering Conference (RE)*, pages 255–264, 2007.

[153] Matthew R. McBride. The software architect. *ACM Communications*, 50(5):75–81, 2007.

[154] Stephen Mellor and Marc Balcer. *Executable UML: A Foundation for Model-Driven Architecture.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[155] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest Editors' Introduction: Model-Driven Development. *IEEE Software*, 20:14–18, 2003.

[156] Matthew B. Miles and A. Michael Huberman. *Qualitative data analysis: an expanded sourcebook*. Sage Publications, 1994.

[157] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[158] Fernando Molina and Ambrosio Toval. Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. *Advances in Engineering Software*, 40(12):1306–1317, 2009.

[159] Francisco Montero and Elena Navarro. ATRIUM: Software Architecture Driven by Requirements. In *14th IEEE International Conference on Engineering of Complex Computer Systems*, pages 230–239, 2009.

[160] John Mylopoulos, Lawrence Chung, and Brian A. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, June 1992.

[161] Eila Niemela and Anne Immonen. Capturing quality requirements of product family architecture. *Information and Software Technology*, 49(11-12):1107–1120, 2007.

[162] Anton Nijholt. *Context-Free Grammars: Covers, Normal Forms, and Parsing*. LNCS. Springer, 1980.

[163] Natalya Fridman Noy and Carole D. Hafner. The state of the art in ontology design: A survey and comparative review. *AI Magazine*, 18:53–74, 1997.

[164] Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

[165] OASIS. Reference Model for Service-Oriented Architecture 1.0, 2006.

[166] Briony J. Oates. *Researching information systems and computing.* Sage Publications, 2006.

[167] Liam O'Brien, Paulo Merson, and Len Bass. Quality Attributes for Service-Oriented Architectures. In *International Workshop on Systems Development in SOA Environments (SDSOA)*, page 3, 2007.

[168] Thomas Olsson, Richard Berntsson-Svensson, and Björn Regnell. Non-functional requirements metrics in practice - an empirical document analysis. In *MeReP*, 2007.

[169] OMG. MARTE UML Profile, Beta 2, 2008. `http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf`.

[170] OMG. UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, v1.1, 2008. `http://www.omg.org/spec/QFTP/1.1/`.

[171] OMG. Semantics of a Foundational Subset for Executable UML Models (FUML), February 2011. `http://www.omg.org/spec/FUML/1.0/PDF`.

[172] Marc Oriol, Jordi Marco, Xavier Franch, and David Ameller. Monitoring Adaptable SOA-Systems using SALMon. In *Workshop on Service Monitoring, Adaptation and Beyond (Mona+)*, 2008.

[173] Ipek Ozkaya, Len Bass, Raghvinder S. Sangwan, and Robert L. Nord. Making Practical Use of Quality Attribute Information. *IEEE Software*, 25(2):25–33, 2008.

[174] Claus Pahl, Simon Giesecke, and Wilhelm Hasselbring. Ontology-based modelling of architectural styles. *Information and Software Technology*, 51:1739–1749, December 2009.

[175] Jose Ignacio Panach, Sergio España, Ana M. Moreno, and Oscar Pastor. Dealing with Usability in Model Transformation Technologies. In *International Conference on Conceptual Modeling (ER)*, pages 498–511, 2008.

[176] Oscar Pastor and Juan Carlos Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling.* Springer, 2007.

[177] Daniel Perovich, Cecilia Bastarrica, and Cristián Rojas. Model-Driven approach to Software Architecture design. In *ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, pages 1–8, 2009.

[178] Shari Lawrence Pfleeger and Barbara A. Kitchenham. Principles of survey research - part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16–18, November 2001.

[179] K. Pohl and C. Rupp. *Requirements Engineering Fundamentals.* Rocky Nook, 2011.

[180] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques.* Springer, 1st edition, 2010.

[181] Eltjo R. Poort, Nick Martens, Inge van de Weerd, and Hans van Vliet. How architects see non-functional requirements: beware of modifiability. In *18th international conference on Requirements Engineering: foundation for software quality (REFSQ)*, REFSQ'12, pages 37–51, Berlin, Heidelberg, 2012. Springer-Verlag.

[182] Samuel Renault, Óscar Méndez Bonilla, Xavier Franch, and Carme Quer. A Pattern-based Method for building Requirements Documents in Call-for-tender Processes. *International Journal of Computer Science & Applications*, 6(5):175–202, 2009.

[183] James Robertson and Suzanne Robertson. Volere. Requirements Specification Template. Edition 15. Technical report, Atlantic Systems Guild, 2010.

[184] Suzanne Robertson and James Robertson. *Mastering the Requirements Process (2nd ed.).* Addison-Wesley Longman Publishing Co., Inc., 2006.

[185] Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers (2nd ed.).* Blackwell Pub., 2002.

[186] Genaína N. Rodrigues, David S. Rosenblum, and Sebastian Uchitel. Reliability prediction in Model-Driven Development. In L. Bri and C. Williams, editors, *8th International Conference on Model Driven Engineering Languages and Systems*, volume 3713, pages 339–354. Springer-Verlag Berlin, 2005.

[187] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, January 2005.

[188] Simone Röttger and Steffen Zschaler. Model-Driven Development for Non-functional Properties: Refinement Through Model Transformation. In *International Conference on the Unified Modeling Language (UML)*, pages 275–289, 2004.

[189] Marcela Ruiz, David Ameller, Sergio España, Pere Botella, Xavier Franch, and Oscar Pastor. Ingeniería de requisitos orientada a servicios: características, retos y un marco metodológico. In *Jornadas de Ciencia e Ingeniería de Servicios (JCIS)*, 2011.

[190] Per Runeson. A Survey of Unit Testing Practices. *IEEE Software*, 23(4):22–29, 2006.

[191] Giedre Sabaliauskaite, Annabella Loconsole, Emelie Engström, Michael Unterkalmsteiner, Björn Regnell, Per Runeson, Tony Gorschek, and Robert Feldt. Challenges in Aligning Requirements Engineering and Verification in a Large-Scale Industrial Context. In *16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 128–142, 2010.

[192] Emila Sadraei, Aybüke Aurum, Ghassan Beydoun, and Barbara Paech. A field study of the requirements engineering practice in Australian software industry. *Requirements Engineering Journal*, 12(3):145–162, 2007.

[193] Carolyn B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.

[194] Mojtaba Shahin, Peng Liang, and Mohammad-Reza Khayyambashi. Architectural design decision: Existing models and tools. In *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (ECSA/WICSA)*, pages 293–296, 2009.

[195] Mary Shaw. The coming-of-age of software architecture research. In *23rd International Conference on Software Engineering (ICSE)*, ICSE '01, page 656, Washington, DC, USA, 2001. IEEE Computer Society.

[196] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline.* Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 1996.

[197] Arnor Solberg, Jon Oldevik, and Jan O. Aagedal. A framework for QoS-aware model transformation, using a pattern-based approach. In R. Meersman, editor, *On the Move Confederated International Workshop and Conference*, volume 3291, pages 1190–1207. Springer-Verlag Berlin, 2004.

[198] Miroslaw Staron. Adopting Model Driven Software Development in Industry - A Case Study at Two Companies. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *9th International Conference Model Driven Engineering Languages and Systems (MoDELS)*, volume 4199 of *Lecture Notes in Computer Science*, pages 57–72. Springer, 2006.

[199] Ashley Sterritt and Vinny Cahill. Customisable Model Transformations based on Non-functional Requirements. In *IEEE Congress on Services (SERVICES)*, pages 329–336. IEEE Computer Society, 2008.

[200] Sam Supakkul and Lawrence Chung. The RE-Tools: A multi-notational requirements modeling toolkit. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 333–334, sept. 2012.

[201] Antony Tang, Muhammad Ali-Babar, Ian Gorton, and Jun Han. A survey of architecture design rationale. *Journal of Systems and Software*, 79(12):1792–1804, December 2006.

[202] Antony Tang, Jun Han, and Rajesh Vasa. Software Architecture Design Reasoning: A Case for Improved Methodology Support. *IEEE Software*, 26(2):43–49, 2009.

[203] Antony Tang, Yan Jin, and Jun Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918–934, 2007.

[204] Susanna Teppola, Päivi Parviainen, and Juha Takalo. Challenges in Deployment of Model Driven Development. In *4th International Conference on Software Engineering Advances (ICSEA)*, pages 15–20, sept. 2009.

[205] TOGAF. The Open Group Architecture Framework Version 9, 2009.

[206] Jeff Tyree and Art Akerman. Architecture decisions: demystifying architecture. *IEEE Software*, 22:19–27, 2005.

[207] Uwe van Heesch and Paris Avgeriou. Naive architecting - understanding the reasoning process of students: a descriptive survey. In *4th European conference on Software architecture*, ECSA'10, pages 24–37, Berlin, Heidelberg, 2010. Springer-Verlag.

[208] Uwe van Heesch and Paris Avgeriou. Mature Architecting - A Survey about the Reasoning Process of Professional Architects. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, volume 0, pages 260–269, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[209] Uwe van Heesch, Paris Avgeriou, and Rich Hilliard. A documentation framework for architecture decisions. *Journal of Systems and Software*, 85(4):795–820, April 2012.

[210] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 2009.

[211] Hiroshi Wada, Junichi Suzuki, and Katsuya Oba. A Model-Driven Development framework for Non-Functional Aspects in Service

Oriented Architecture. *International Journal of Web Services Reseseach*, 5(4):1–31, 2008.

[212] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical Research Methods in Software Engineering. In Reidar Conradi and AlfInge Wang, editors, *Empirical Methods and Studies in Software Engineering*, volume 2765 of *Lecture Notes in Computer Science*, pages 7–23. Springer Berlin Heidelberg, 2003.

[213] Robert K. Yin. *Case Study Research: Design and Methods (4th ed.)*. Sage Publications, 2009.

[214] Eric Yu, Paolo Giorgini, Neil Maiden, and John Mylopoulos. *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[215] Norazlin Yusop, Didar Zowghi, and David Lowe. The Impacts of Non-Functional Requirements in Web System Projects. *International Journal of Value Chain Management*, 2(1):18–32, 2008.

[216] John A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292, 1987.

[217] Liming Zhu and Ian Gorton. UML Profiles for Design Decisions and Non-Functional Requirements. In *ICSE workshop*, page 8, 2007.

[218] Liming Zhu and Yan Liu. Model Driven Development with Non-Functional Aspects. In *15th Workshop on Early Aspects/ICSE Worshop on Aspect-Oriented Requirements Engineering and Architecture Design*, pages 49–54, 2009.

Index