Vol. 1 | No. 1 | 2013

SciKA

International Journal of Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

IJISPM

005

Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement

Björn Johansson
Markus Lahtinen

019

A multi-layered software architecture model for building software solutions in an urbanized information system

Sana Guetat
Salem Dakhli

037

A case study on variability management in software product lines: identifying why real-life projects fail

Tom Huysegoms
Monique Snoeck
Guido Dedene
Antoon Goderis
Frank Stumpe

SciKA - Association for Promotion and Dissemination of Scientific Knowledge

SciKA

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

Mission

The mission of the IJISPM - International Journal of Information Systems and Project Management - is the dissemination of new scientific knowledge on information systems management and project management, encouraging further progress in theory and practice.

The IJISPM publishes leading scholarly and practical research articles that aim to advance the information systems management and project management fields of knowledge, featuring state-of-the-art research, theories, approaches, methodologies, techniques, and applications.

The journal serves academics, practitioners, chief information officers, project managers, consultants, and senior executives of organizations, establishing an effective communication channel between them.

Description

The IJISPM offers wide ranging and comprehensive coverage of all aspects of information systems management and project management, seeking contributions that build on established lines of work, as well as on new research streams. Particularly seeking multidisciplinary and interdisciplinary perspectives, and focusing on currently emerging issues, the journal welcomes both pure and applied research that impacts theory and practice.

The journal content provides relevant information to researchers, practitioners, and organizations, and includes original qualitative or qualitative articles, as well as purely conceptual or theoretical articles. Due to the integrative and interdisciplinary nature of information systems and project management, the journal may publish articles from a number of other disciplines, including strategic management, psychology, organizational behavior, sociology, economics, among others. Articles are selected for publication based on their relevance, rigor, clarity, novelty, and contribution to further development and research.

Authors are encouraged to submit articles on information technology governance, information systems planning, information systems design and implementation, information technology outsourcing, project environment, project management life-cycle, project management knowledge areas, criteria and factors for success, social aspects, chief information officer role, chief information officer skills, project manager role, project manager skills, among others.

Topics covered

The journal offers comprehensive coverage of information systems management and project management.

The topics include, but are not limited to:

▪ information technology governance	▪ project environment	▪ project management knowledge areas
▪ information systems planning	▪ project management life-cycle	▪ scope management
▪ information systems design and implementation	▪ project initiation	▪ time management
▪ information technology outsourcing	▪ project planning	▪ cost management
▪ enterprise architecture	▪ project execution	▪ quality management
▪ information systems governance	▪ project control and monitoring	▪ procurement management
▪ information systems department	▪ project closing	▪ risk management
▪ chief information officer role	▪ criteria and factors for success	▪ communication management
▪ information technology leadership role	▪ project manager role	▪ human resources management
▪ chief information officer skills	▪ project manager skills	▪ performance teams
▪ information systems management tools	▪ portfolio management	▪ social aspects
▪ management of complex projects	▪ program management	▪ conflict management
▪ audits	▪ managing organization – structure	▪ managing organization – responsibilities
▪ innovation	▪ tools and techniques	▪ project management office
▪ ethics	▪ project evaluation	▪ Contracts

Special issues devoted to important specific topics will be evaluated for publication.

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

Submissions

Researchers and practitioners are encouraged to submit their manuscripts to the IJISPM. The guidelines for submission can be found at the journal’s home page: www.sciencesphere.org/ijispm

Special issues

Proposals for special issues should be submitted to the Editor-in-Chief. E-mail: editor.ijispm@sciencesphere.org

Advertising information

The journal accepts advertising in the following categories: IT/IS events; IT/IS training and education; IT/IS entities. For full details please contact the editorial office. E-mail: office.ijispm@sciencesphere.org

Correspondence and questions

All correspondence and questions should be directed to João Varajão (Editor-in-Chief). E-mail: editor.ijispm@sciencesphere.org

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Table of contents

SPECIAL FEATURES

1 Editorial
João Varajão, University of Trás-os-Montes e Alto Douro, Portugal

RESEARCH ARTICLES

5 Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement
Björn Johansson, Lund University, Sweden
Markus Lahtinen, Lund University, Sweden

19 A multi-layered software architecture model for building software solutions in an urbanized information system
Sana Guetat, Le Mans University, ARGUMANS Lab., France
Salem Dakhli, Paris-Dauphine University, CERIA Lab., France

37 A case study on variability management in software product lines: identifying why real-life projects fail
Tom Huysegoms, Faculty of Business and Economics, Information Systems Group, KULeuven, Belgium
Monique Snoeck, Faculty of Business and Economics, Information Systems Group, KULeuven, Belgium
Guido Dedene, Faculty of Business and Economics, Information Systems Group, KULeuven, Belgium
Antoon Goderis, KBC ICT, KBC Global Services, Belgium
Frank Stumpe, KBC ICT, KBC Global Services, Belgium

IJISPM

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

Editorial

It is our great pleasure to bring you the first number of the IJISPM - International Journal of Information Systems and Project Management.

The mission of the IJISPM is the dissemination of new scientific knowledge on information systems management and project management, encouraging further progress in theory and practice.

In this inaugural issue, readers will find important contributions by several internationally renowned and experienced researchers, briefly described below.

As Björn Johansson and Markus Lahtinen state in their article "Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement", IT procurement represents a business process of high importance, including the ability to articulate requirements that the procurement deals with. Furthermore, specifying requirements is of importance for both procurer and potential supplier, as it functions as the central contractual element between the two. The purpose of this article is two-fold: (i) to show how established terminology for requirement specification is represented in current call for bids for the procurement of IT; and (ii) to introduce an organizing framework that may assist procurers in actively addressing functional requirements and business requirements. Ten "call for bids" were examined from a Swedish national procurement database. From the analysis of the bids, it can be concluded that: (i) the call for bids displays a high degree of precision regarding hardware aspects, but less precision regarding software; (ii) supplier experience and competence is stressed, but rarely elaborated on in detail; and (iii) call for bids vagueness may be used as a lock-in opportunity for suppliers. From the discussion on this, a tentative procurement framework is suggested, aiming on increasing the logical transparency for the procurement of IT.

The second article "A multi-layered software architecture model for building software solutions in an urbanized information system" is co-authored by Sana Guetat and Salem Dakhli. The concept of Information Systems urbanization has been proposed since the late 1990's in order to help organizations building agile information systems. Nevertheless, despite the advantages of this concept, it remains too descriptive and presents many weaknesses. In particular, there is a lack of useful architecture models dedicated to defining software solutions compliant with information systems urbanization principles and rules. Moreover, well-known software architecture models do not provide sufficient resources to address the requirements and constraints of urbanized information systems. In this paper, the authors draw on the "information city" framework to propose a model of software architecture - called the 5+1 Software Architecture Model - which is compliant with information systems urbanization principles and helps organizations in building urbanized software solutions. This framework improves the well-established software architecture models and allows the integration of new architectural paradigms. Furthermore, the proposed model contributes to the implementation of information systems urbanization in several ways. On the one hand, this model devotes a specific layer to applications integration and software reuse. On the other hand, it contributes to the information system agility and scalability due to its conformity to the separation of concerns principle.

Economies of scale can be seen as some kind of "holy grail" in state of the art literature on the development of sets of related software systems. Software product line methods are often mentioned in this context, due to the variability management aspects they propose, in order to deal with sets of related software systems. Both variability management and software product lines already have a strong presence in theoretical research, but in real-life software product line projects trying to obtain economies of scale still tend to fall short of target. The objective of the third paper, "A case study on variability management in software product lines: identifying why real-life projects fail", presented by Tom Huysegoms, Monique Snoeck, Guido Dedene, Antoon Goderis and Frank Stumpe, is to study this gap between theory and reality through a case study in order to analyze why such a gap exists, and to find a way to bridge it. Through analysis of the causes of failure identified by the stakeholders in the case study, the underlying problem, which is found to be located in the requirements engineering phase, is crystallized. The identification of a framework describing the problems will provide practitioners with a better focus for future endeavors in the field of software product lines, so that economies of scale can be achieved.

We would like to take this opportunity to express our gratitude to the distinguished members of the Editorial Board, for their commitment and for sharing their knowledge and experience in supporting the IJISPM.

Finally, we would like to express our gratitude to all the authors who submitted their work, for their insightful visions and valuable contributions.

We hope that you, the readers, find the International Journal of Information Systems and Project Management an interesting and valuable source of information for your continued work.

The Editor-in-Chief,

João Varajão

University of Trás-os-Montes e Alto Douro

Portugal

João Varajão is a professor of Information Systems Management, Project Management and Software Engineering at the *University of Trás-os-Montes e Alto Douro* and a visiting professor at the *University of Porto Business School*. He is also a researcher of the *Centro Algoritmi* at the *University of Minho*. Born and raised in Portugal, he attended the *University of Minho*, earning his Undergraduate (1995), Masters (1997) and Doctorate (2003) degrees in Technologies and Information Systems. In 2012, he received his Habilitation degree from the *University of Trás-os-Montes e Alto Douro*. His current main research interests are in Information Systems Management and Project Management. Before joining academia, he worked as an IT/IS consultant, project manager, information systems analyst and software developer, for private companies and public institutions. He has supervised more than 50 Masters and Doctoral dissertations in the Information Systems field. He has published over 250 works, including refereed publications, authored books, edited books, as well as book chapters and communications at international conferences. He serves as editor-in-chief, associate editor and member of the editorial board for international journals and has served in numerous committees of international conferences and workshops. He is co-founder of CENTERIS - Conference on ENTERprise Information Systems.

www.shortbio.net/joao@varajao.com

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement

Björn Johansson
Department of Informatics, Lund University
Ole Römers väg 6, Lund SE-223 63
Sweden
www.shortbio.net/bjorn.johansson@ics.lu.se

Markus Lahtinen
Department of Informatics, Lund University
Ole Römers väg 6, Lund SE-223 63
Sweden
www.shortbio.net/markus.lahtinen@ics.lu.se

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

B. Johansson and M. Lahtinen, “Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement,” *International Journal of Information Systems and Project Management*, vol. 1, no. 1, pp. 5-16, 2013.

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

Getting the balance right between functional and non-functional requirements: the case of requirement specification in IT procurement

Björn Johansson
Department of Informatics, Lund University
Ole Römers väg 6, Lund SE-223 63
Sweden
www.shortbio.net/bjorn.johansson@ics.lu.se

Markus Lahtinen
Department of Informatics, Lund University
Ole Römers väg 6, Lund SE-223 63
Sweden
www.shortbio.net/markus.lahtinen@ics.lu.se

Abstract:
IT procurement represents a business process of high importance, including the ability to articulate requirements that the procurement deals with. Furthermore, specifying requirements is of importance for both procurer and potential supplier, as it functions as central contractual element between the two. The purpose of this article is two-fold: *(i) to show how established terminology for requirement specification is represented in current call for bids for the procurement of IT; and (ii) to introduce an organizing framework that may assist procurers in actively addressing functional requirements and business requirements.* Ten "call for bids" were examined from a Swedish national procurement database. From the analysis of the bids, it can be concluded that: (i) the call for bids displays a high degree of precision regarding hardware aspects, but less precision regarding software; (ii) supplier experience and competence is stressed, but rarely elaborated on in detail; and (iii) call for bids vagueness may be used as a lock-in opportunity for suppliers. From the discussion on this, a tentative procurement framework is suggested, aiming on increasing the logical transparency for the procurement of IT.

Keywords:
procurement of IT; software specification; government procurement; software requirement; procurement checklist.

DOI: 10.12821/ijispm010101

Manuscript received: 18 December 2012
Manuscript accepted: 4 February 2013

1. Introduction

For firms and public administration, procurement represents a business process of high importance. Regarding the role of Information Technology (IT), it may be used as a tool to increase an organizations' procurement capability [1, 2] and IT may also be the subject of the procurement. In this paper we are interested in the latter and in particular the case of the European Union-wide legislation of "Government procurement in the European Union". Put shortly, the legislation stipulates - and in line with the ambitions of the common market - that an individual Government procurement exceeding a certain monetary threshold value needs to be published in open competition. The act applies to all services and goods that government bodies procure, including the procurement of IT.

From an academic point of view, procurement in a broad sense has previously been covered extensively [3]. Less interest has been given to procurement of IT. In the majority of call for bids, financial value of the bid is not specified by the procurer, because the supplier is required to submit a proposal of the financial value. However, in Sweden 2009 the part of call for bids that pre-specified value of procurement of IT amounted to 63 million Euro. Further illustrating this trend towards increased importance of procurement is the increased amount of job postings explicitly demanding purchasing competence surrounding procurement of IT. Also, Swedish trade press has on numerous occasions reported on challenges associated with procurement legislation. Rådmark [4] reports on the CIO of Växjö Municipality having to resign as a result of not complying with Government procurement in the European Union. Eriksson [5] and Eriksson [6] comments upon on procurement challenges when procuring vaccine for the Health sector and procuring services with Swedish Civil aviation is literally described as being a "nightmare". Similar challenges have been reported by Du Preez [7] from the UK. In its capacity of overseeing the efficiency and effectiveness of the civil services in the UK, the Cabinet Office has ordered a review of current Information and Communication Technologies (ICT) procurement procedures as direct response to the reported lack of supplier scope in Government IT procurement. Put shortly, Small and Medium Enterprises (SME) are suggested to be cut out in the bidding process by larger enterprises. Conspicuously and somewhat ironically, the same Cabinet Office [8] offered an ICT Strategy Implementation manual under the parole "Moving from the 'what' to the 'how'" the past year. We believe it is important to address procurement of IT, not only because of the significant financial value that it represents, but also to address the reported challenges regarding compliance.

The ability to specify requirements is another important area for firms and public administration when they aim at equipping themselves with IT-based resources. This implies the need to specify requirements for future software to a satisfactory degree. Furthermore, being able to specify requirements becomes important since organizations increasingly buy the software as either standardized software package or buy the software development as a consultancy service rather than develop them in-house.

Pivotal to any activity of system development analysis and design is the process of specifying requirements [9]. Also, as stated above, requirement specification process applies to instances of both in-house development, as well as instances of procuring IT from external suppliers.

Consequently, we argue that requirements may be used as an anchor to analyze a procurement situation. Using call for bids, the purpose of this article is two-fold: *(i) to show how established terminology for requirement specification is represented in current call for bids for the procurement of IT; and (ii) to introduce an organizing framework that may assist procurers in actively addressing functional requirements and business requirements.* In order to reach the purpose, the following question was initially asked: How are functional and non-functional requirements represented in current call for bids for the procurement of IT?

The rest of the paper is organized as follows: we first present some descriptions and problematic issues with systems development and requirements focusing specifically on software requirements specifications. The section thereafter presents our research method, how empirical data were collected and analyzed. In the penultimate section we then present our findings which we discuss and draw some conclusions in the final section.

2. Systems development and requirements

A major problem in Information Systems (IS) development is misalignment between needed functionality and the functionality offered in the developed IS. This could be stated as being a continuous challenge - independent on if it is internal development or if it is procurement of standard software package. This problem could be described as the distance between what end-users want to have support for in the business processes, and what the IS de facto gives support for. There are definitely a lot of different reasons for why this is the case. But, it can be stated that one important consideration are difficulties of translating and transferring business requirements from identification to specification, and further into implementation.

To have some input on this it is first important to stipulate what requirements and especially what (process) business requirements are. Jackson [9] propose that requirements are descriptions of the application domain and the problems to be solved, and sees the challenge in the requirement collection process between the method for problem structure on one hand and the description per se on the other. The same is described by Power [10] who speaks about “requirements as needs” and “requirements as text”. Both authors emphasize the distinction between these two different types of requirements specifications. They describe the challenge of transforming requirements from needs to text and into formal descriptions, which can be easily transformed into program code and software system features.

Jackson [9] makes a distinction between requirements and specifications and states that specifications are descriptions of the interface between the developed system and the application domain. This is in line with the statement that a requirement specification should form a bridge between requirements engineering and software engineering [11]. A brief investigation of what requirements are could suggest that requirements are clear and well-defined. But, that is not the case, and there are several reasons for that. There have been attempts both from research and practice to classify and categorize requirements, resulting in classification schemes that distinguish between functional and non-functional requirements [10]. For example, the IEEE standard for the software requirements specification [12] distinguishes fourteen types of requirements, divided into functional requirements and thirteen types of non-functional requirements. Robertson and Robertson [13] make a similar distinction when they describe seventeen different types of requirements divided into product constraints, functional requirements and non-functional requirements. From this it can be suggested that requirements could be seen as either: a function, capability, or property of a proposed system; and/or, the statement of such a function, capability, or property [10] and/or as described by Jackson [9] as descriptions of the application domain and the problems to be solved there. This last description emphasizes on what and not how. This is to some extent in conflict with the description from Zave and Jackson [14] that state: there was a time when the epigram “requirements say what the system will do and not how it will do it” summarized all of requirements engineering. That time is long past.

The statement by Zave and Jackson [14] could be interpreted as being a change in what requirements should describe has occurred and that requirement specification now also focus on how the developed system will execute the wanted requirement. This statement suggests that the scope of what requirements are have broadened over time.

Literature on software development often makes a high-level distinction between functional and non-functional requirements (cf. RUP, software engineering). The software engineering-approach to software specification has not gone without criticism; Odeh and Kamm [15] state that the formalism associated with Unified Modeling Language (UML) techniques are not suitable for translation of business models into software models. Technical methodology, by its very nature, does not take organizational aspects into account, for example dynamic internal political agendas and conflicting interests and interpretations among the involved stakeholders. Nevertheless, and returning to the high-level separation, functional requirements represent the type of operations that connects the user and problem domain with the representation of the problem domain [16]. More specifically, functional requirements may be divided into four major categories of operations: calculation, signaling, update and write. According to Stellman and Greene [17] non-functional requirements represent requirements beyond the above mentioned; for example usability, computing efficiency, reliability, scalability, reusability, portability, etc. In tandem with specified requirements, use cases define interaction with a suggested system from an actor's point of view. In all of the above mentioned categories, the requirements need to be specified, clarified and documented in some way. The result of this exercise often ends in a software requirements specification (SRS).

Software requirements specifications are important documents, used by different groups of people for communicative purposes; by customers, to know what to expect; by the software developers, to know what to build and how; by test groups, to test and evaluate the system [18-20]. The SRS act as a channel of communication between developers and customers and help to ensure that the system satisfies customer needs [21]. Moreover, it creates a baseline upon which sub-sequent systems development activities are based [22].

It is clear that an SRS is one part of the overall systems requirements determination process which in its turn is part of the entire systems development process. An SRS is described by Eriksson [23] as a document produced when a system is built from scratch, or if there are major changes being made to an existing system. Wiktorin [24] on the other hand states that "a requirements specification consists of several parts". Another description, also rather short, is given by Duggan and Thachenkary [25]: "Requirements specification: representing the results [of the previous steps in the SRD process] in a document".

One explanation of the contents of an SRS is given by Wieringa [26], who states the following: "A requirements specification consists of a specification of product objectives and a specification of required product behavior".

In other words, an SRS shows the purpose of the system, and how it is supposed to behave - its functionality, which is described by Carvalho et al. [27] in the following way: an SRS should describe the "what" of a system, not the "how". Wiegers [18] states that since the SRS is important for the following activities in systems development, it needs to have a detailed description of system behavior. Smith et al. [28] state that the SRS should describe essential system requirements of the software and its external interfaces, such as functions, performance, constraints and quality attributes. Another similar description of the SRS is given by IEEE in standard 12207: "the systems requirements specification shall describe: functions and capabilities of the system; business, organizational and user requirements; safety, security, human-factors engineering (ergonomics), interface, operations, and maintenance requirements; design constraints and qualification requirements" [29]. Due to its proximity to spoken language, the natural language approach to SRS is arguably the most common one [22]. While natural language may suffice in very informal and small software requirement specification situations, they represent an exception since precision and formality is not a priority. However, since more stakeholders get involved and contractual relationship becomes a reality, the natural language approach does not suffice for several reasons. For example, natural language does not lend itself to be efficiently coordinated and communicated in a team of stakeholders - this could be further problematic due to geographical and cultural asymmetries. Finally, according to Daniels and Bahill [30] highly complex systems require a higher level of formality.

To sum up, a SRS is a document created when a system is built or rebuilt, containing purpose and behavior of the system as well as descriptions of the system and its desired functions. As have been discussed above and depending on the analytical approach to software requirement specification, the degree of precision and formality in requirements vary. More technical approaches generally excludes important organizational aspects that impacts requirement specification. The softer approach - mostly accentuated by the natural language approach loses out on communicative precision and formality. Finally, not enough theoretical attention has been directed towards the increasing use of handling requirement specification contractually. Thomsen [31] has contributed with an elaboration on the meaning of purchasing competence in relation to IS, which is of particular importance in procurement situations. But, requirement specification lies outside the analysis scope. Consequently, there is a need to explore the use of requirement specification in the case of contractual relationship. Given our review above, in the context of this paper we view the 'call for bids' as a sort of initial formalization of a SRS nonetheless. This further allows us for being open from an interpretative point of view towards our empirical material.

3. Data collection and collected data

Ten current calls for bids concerning the procurement of IT were reviewed (see Table 1). We attempted to generate observations, primarily based on a data-driven approach, to problematize our theoretical understanding. The ten cases were hard-copy printed in completeness in two sets. Supplied with post-it notes, it was decided that we (the two authors of the paper), independently from each other, read through the call for bids looking for theoretical gaps in terms of functional and non-functional requirements. The two sets of post-it notes were then compared and discussed. This exercise ended in some findings and the analytically most promising were selected for future analysis. The selection was made from the question asked: How are functional and non-functional requirements represented in current call for bids for the procurement of IT?

More specifically, we were interested in assessing both use as well as the usefulness in relation to how the 'call for bids' were formulated. The selected calls for bids were collected from e-Avrop, which is Sweden's biggest free database for procurement.

Table 1. Description of the 10 call for bids cases

Case	Subject of procurement	Categorization of the subject in the database e-Avrop
Örnsköldsvik Municipality	New external webpage	Production of a new external webpage for the municipality in accordance with pre-specified graphical profile program
Östra Göinge Municipality	Procurement of surf pads	120-130 units of tablet computers
Eskilstuna Municipality	Web-based support system for relatives of patients	Web-based turn-key support system for relatives of patients
Umeå University	Video conferencing system	Video conferencing hardware
Sundsvall Municipality	Unspecified IT	Experience output-specified
Courts of Sweden	Service and support for a video conferencing system	Service and support for video conferencing system
Flen Municipality	Creative Media	IT-based education on creative media main focus.
Sundsvall Municipality	Apple-products	Pre-procurement of tablet computers
Örnsköldsvik Municipality	Surf pads	24 units of tablet computers
Skellefteå Municipality	SMS platform	Meta-integration of disparate SMS-systems delivered as Software as a Service

4. Data collection and collected data

In this section we present findings from the analysis of the call for bids. The first finding discussed is the different precision between hardware and software requirements. Next finding discussed is the demand for a track record of the bidders from the procurer and how that is related to low specificity of requirements. This finding is closely related to the third finding discussed which discusses how low specificity restricts or gives opportunities for suppliers. The last finding discussed is the inherent tension existing between functional/non-functional requirements on the one hand and hardware/software on the other hand. From the discussion on these four findings we present a procurement framework which we suggest could act as a starting point for developing a strategy for the procurement of IT.

4.1 Hardware precision and software vagueness

Three of the cases concerned procurement of tablet computers. Sundsvall and Örnsköldsvik Municipality are pure hardware purchases, while the procurement of Östra Göinge Municipality stipulates tablet computers to be used in pre-school and compulsory school. Consequently, the suggested user group can be assumed being pupils in the age span 6-16 years. Usability requirements are also included in the call for bids.

The requirement specification of Östra Göinge Municipality shows a high level of precision regarding hardware and product specification and less in terms of detailed software performance. The high level of hardware and product specification is illustrated by the following three examples:

*"The pad must have a built-in battery providing at least 8 hours of battery time. [...] The pad must be delivered with a minimum of 16 Gb primary memory. [...] The pad must have a minimum screen of 9,5" allowing for 1024*768 resolution" (Östra Göinge Municipality, 2011a)*

As is common in call for bids, the overarching evaluation criterion are given weights, and in this case price quality is given 5/10 of evaluation importance, "usability" is given 4/10 of importance and insurance solution is given 1/10 of importance. As the call for bids is formulated, hardware aspects are included in the usability definition (see point 8 below) and the generic minimum performance level of software is not covered in any detail. According to Östra Göinge Municipality (2011b, p. 4), usability will be assessed based on the quality of the following aspects: 1) Protective cover; 2) File management; 3) Application management (purchasing, installation and remove); 4) Administrative tools; 5) Security back-up; 6) Boot-up time; 7) Experience of performance (lagging, efficiency in switching between windows); and 8) Hardware.

Besides the hardware requirements previously mentioned before, the criterion ranging from 1-7 are not expressed in any measurable minimum metrics. Due to context-dependent quality of usability, it has been suggested [32-34] that usability should be expressed in measurable context-specific metrics. For example, boot-up time may be specified in seconds. In the Östra Göinge case the criteria is supposed to be evaluated subjectively by the procurement staff. Further improvement would have allowed for evaluation to be made by stakeholders that are supposed to use the pads, if not, any objective measurement such as seconds could be used.

This is a call for a more coherent use of non-functional requirements. It can also be stated from the findings of the Östra Göinge case that there exists possibilities for procurers to be more precise when describing how the evaluation is planned to be done as well as what evaluation criteria that is supposed to be used. The benefits for being more precise would be two folded. First the bidders would be able to more clearly evaluate their products and thereby prevent errors, as well as being able to give a more accurate price. Second, the procurers' evaluation of specific proposals would be much more easy to conduct.

4.2 The objects looking for a subject or the requirements looking for a supplier

In procurement processes we have identified numerous instances of explicit demand by the procurer for the supplier to prove a track-record of previous deliveries. This illustrates the importance of the subject, meaning the supplier. In the process of requirement specification, which could be seen as the object of the activity, the screened "call for bids" clearly states the qualities the suppliers need to have. We can only speculate on implications of this, it could be used as a screening-mechanism of supplier, guaranteeing that the supplier has been able to deliver in the past. However, the obvious risk is that more competitive suppliers are excluded from the bidding process.

The qualification of a supplier is given in a rather superficial way. One example of this could be found in the bid from Örnsköldsvik Municipality in which it is stated “The provider shall have required experience and competence to deliver and be a provider that has delivered similar services before”. The implications from this requirement could mean difficulties both from suppliers’ perspective as well as from buyers’ perspective. For the suppliers it means that they have to make a statement on its experience and competence, which could be hard if recently established. It also means that the providers need track-records that are positive and to have those they probably need to have been in the business for some time. For the buyer it could be seen as strictly positive to get references from earlier customers to the provider and it is probably easier to evaluate a provider if the provider could present a track record of successful deliveries. However, for both the supplier and the buyer, this requires a satisfactory level of articulation of both experience and competence.

The theoretical implication following from this is that the model of functional and non-functional requirements does not adequately address the issue of “who”, suggesting that further research activities need to be focused on including qualities of the supplier into the existing models. Considering the importance given in investigated call for bids, it is clear that more consideration needs to be taken to include qualities of the suppliers to have a useful framework for procurement of IT.

4.3 Vagueness as a supplier opportunity

From theoretical point of view the call for bids are fuzzy in terms of precision on the absolute-haves. In the case of Eskilstuna Municipality under “Requirements for the service”, of the presented six bullets regarding requirements relating to the service in itself were five fuzzy in terms of how to evaluate them. The sixth requirement, “being able to link externally” to the service we find redundant since external linking to web service in most cases always are possible. Under the same headline there are numerous of shall requirements, however, the evaluation of all these requests are handed over to the supplier. The suppliers are asked to state if each and every of the requirements are fulfilled by giving a yes or no answer on the direct question “is the requirement fulfilled?” Potential suppliers are also asked to make comments on each answer. This may be a satisfactory way of having a supplier to make the first evaluation of the bid. However, it demands clear and precise requirements, as well as a clear description of what the procurers wants to have. In the Eskilstuna case the call for bids is fuzzy in the description of what is asked for, which makes that potential suppliers have an opportunity to also be fuzzy in their bids. Thereby rendering the supplier an opportunity to clearly state that they fulfill the requirements.

Another case that epitomizes vagueness as a supplier opportunity is the Sundsvall Municipality case on “unspecified IT”. The case presents Sundsvalls Municipality’s vision about their new building at the big square and asks the supplier to deliver something that makes visitors to be so extremely surprised that it creates an “Oh, shit”-feeling (literal translation) among the procuring party. This call for bids is a clear example on how an organization uses suppliers to help them create innovative solutions. However, it also is a clear example on how suppliers could use the vagueness to try out some innovations and at the same time give them a possibility to get a “big project” if they want that. The call for bids is relatively thin and it does not say anything about the level or scope - in the form of needed or available resources - of the project, which also indicates suppliers’ opportunity.

4.4 Procurement framework – strategy for the procurement of IT

In the studied cases we have observed a lack of explanation of what “product” that was asked for, how the “product” should be used, as well as who should use the “product”. This could be improved by using use cases when specifying requirements in the procurement of IT. Furthermore, non-functional requirements include a multitude of possible use qualities that may be connected to software, hardware and also requirements beyond the scope of hardware and software. In the circumstance of IT procurement, we argue for more purposeful representations of IT to assist procurement staff in organizing requirements. In line with this, it is necessary to address the inherent tension existing between functional/non-functional requirements on the one hand and hardware/software on the other hand. Several of the studied cases conflate hardware/software requirements with functional and non-functional requirements, creating logical inconsistencies since the realization or delivery of non-functional requirements needs to consider the interactive nature between hardware and software. In concrete, a call for bid which specifies hardware to restrictive might negatively impact the realization of functional and non-functional requirements.

From the findings above, we argue that there exist limitations when only considering requirements in terms of functional and non-functional requirements due to wide scope of what may be included under the umbrella term of non-functional requirements. In addition, from the call for bids we have shown that from a user point of view - the separation between functional and non-functional requirements is problematic.

One way forward to resolve some of the above mentioned problems could be to introduce the product model suggested by Kotler and Keller [35]. In our view this model represents an alternative way of separating functional and non-functional requirements, but also to clarify both functional as well as non-functional requirements. In particular, we hypothesize that merging requirement terminology with the product layers suggested by Kotler and Keller [35] is a more useful way to specify requirements under practical procurement activities. Separating the instance of IT being procured into core product (the essential problem-solving side of a product), basic product (the actual product, e.g. SW and HW), expected product (what the customer expect the product to include), augmented product (attributes beyond the scope of the actual product, e.g. insurance, guarantees and deliver times) and potential product (the qualities important for the future use of the product) assists procurers to make increasingly sense of the separation between functional and non-functional requirements. In particular this holds true for non-functional requirements, due to unclear scope of what quality attributes to include as non-functional requirements.

Placing the different product layers alongside the requirement terminology (Table 2) contextualizes the requirements into a wider business-driven framework more useful for procurement activities.

Table 2. Matching product levels with requirement terminology

Kotler and Keller [35])	**Suggested requirement specification equivalent**	**Comments**
Potential product	Scalability, Flexibility, Reusability, “Nice to have in the future”	The requirements dealing with dynamic requirements that changes over time, for example increased users
Augmented product	Guarantees, service and maintenance contracts, “small extras”. Non-functional requirements, e.g. availability, portability, integrity, reliability, reusability, robustness	Augmented product overlaps mostly with non-functional requirements, e.g. availability, portability, integrity, reliability, robustness. Performance is considered a non-functional requirements, but is included in the scope of usability efficiency (cf. [34]; [33]; [32])
Expected product	System objectives expressed in terms of assumed improvement organizational benefits, i.e. the business-realizing of requirements	In order to cover the business-side of requirements in the framework, and to fulfill the customer needs (cf.[21]), the expected products should be expressed as technology-neutral desired outputs
Basic product	Use cases, usability	Use cases in combination with usability considerations provide a language to address the theoretically desired quality of what rather than how (cf. [26];[27])
Core benefit	Customer value/citizen utility	Essentially the choice of the procurers; and policy-makers, ultimately, in the case of government procurement
Potential product	Scalability, Flexibility, Reusability, “Nice to have in the future”	The requirements dealing with dynamic requirements that changes over time, for example increased users

5. Overarching conclusions

Due to changes in the legal system, public and governmental procurement is increasingly important within the European Union. In line with the ambitions of creating a common European market where suppliers across the union may bid for any call for bids posted publicly. This is assumed to increase competition. Another reported citizen benefit is the increased transparency following from the need for procurers to articulate and specify their need. Theoretically, citizens are given the possibility to scrutinize how tax-payers money is spent. However, challenges and hurdles have been reported. One example is the difficulties non-established firms have in winning a call for bids. Another example is the increased administrative burden that is put on the procuring organization. In this paper we set out to explore 10 call for bids by Swedish municipalities and government bodies. Call for bids can be seen as a form of requirement specification and of special interest in this paper was to examine how requirement specification terminology was expressed and formulated in these call for bids.

From the analysis of the call for bids it can be concluded that non-functional requirements do not sufficiently separate between the business-side and HW/SW-side of the "product" that the municipalities demand in their bids. This conclusion made us search for other ways of specifying requirements in call for bids. One solution on the problematic issue of specifying and separating functional and non-functional requirements could be to use the Kotler and Keller [35] product model.

Furthermore, the analysis suggests that dividing requirements into functional and non-functional requirements results in low precision of requirements as well as limitations in applicability. It is also found that supplier qualities, or the question of "whom", is empirically important in the bids.

Procurers claim to require "functionality" and "usability", however these requirements are rarely expressed in any meaningful level of detail. While this relationship may enable for mutual discussion on what is the most purposeful solution, there is also a financial risk that the procurer takes by the imprecision that can be used both on the margin-side as well as proprietary opportunities side by the bidders. This supports the conclusion that there needs to be a balance between precision and impreciseness, in the call for bids.

Finally, the analysis of the findings of hardware precision and software vagueness it can be concluded that there is a need for an increased coherency of non-functional requirements. It can also be concluded that that procurers have a possibility to be even more precise in the evaluation criteria by for instance working with measurable metrics. In turn, this would potentially make evaluation increasingly efficient, and most likely improve the result of the procurement.

Due to the above reported findings, another focus on how to specify requirements is needed for procuring entities. We suggest that a framework building on Kotler and Keller [35] product model may be more useful and comprehensive for procurers than relying on the separation between functional and non-functional requirements or use cases solely. Properly used, the framework forces the procuring entity to think actively on possible future changes to the identified requirements, for example by acknowledging the importance of scalability and the "nice to haves". While precise functionality regarding software is necessary, it is equally important that a call for bids includes comments on the value it is supposed to generate for the procuring organization. This is also included in the framework. It is important to note that we are not suggesting that the framework is to be used as a substitute for established terminology in systems development. It should rather be seen as an organizing and contextualizing framework that puts software and hardware specifications into a value-adding context.

References

[1] Business Link. (2011). Overview on selling to government: e-procurement [Online]. Available: http://www.businesslink.gov.uk/bdotg/action/detail?itemId=1073792572&type=resources

[2] M. J. Garrido, A. Gutiérrez and R. San Jose, “Organizational and economic consequences of business e-procurement intensity,” *Technovation*, no. *28*, pp. *615-629*, 2008.

[3] M. Rolfstam, W. Phillips and E. Bakker, “Public Procurement of Innovation Diffusion: Exploring the Role of Institutions and Institutional Coordination,“ Centre for Innovation, Research and Competence in the Learning Economy (CIRCLE), working paper, WP 2009/07, 2009.

[4] H. Rådmark. (2011). Växjös kritiserade it-chef slutar [Online]. Available: http://www.idg.se/2.1085/1.405870/vaxjos-kritiserade-it-chef-slutar

[5] P. Eriksson. (2011). Det är LOU som är problemet [Online]. Available: http://www.idg.se/2.1085/1.405813/det-ar-lou-som-ar-problemet

[6] P. Eriksson. (2011). En mardrömslik upphandling [Online]. Available: http://www.idg.se/2.1085/1.405813/det-ar-lou-som-ar-problemet

[7] D. Du Preez. (2012). Cabinet Office puts all ICT procurements on hold [Online]. Available: http://www.cio.co.uk/news/3408309/cabinet-office-puts-all-ict-procurements-on-hold/

[8] Cabinet Office. (2012). Government ICT Strategy - Strategic Implementation Plan [Online]. Available: http://www.cabinetoffice.gov.uk/content/government-ict-strategy-strategic-implementation-plan

[9] M. Jackson, *Software requirements & Specifications: a lexicon of practice, principles and prejudices*, London, UK: ACM Press, 1995.

[10] N. M. Power, *A grounded theory of requirements documentation in the practice of software development*, Dublin, School Dublin City University, pp. *223*, 2002.

[11] M. Jackson, “The meaning of requirements,” *Annals of Software Engineering*, no. *3*, pp. *5-21*, 1997.

[12] IEEE, *IEEE STD 830-1998: IEEE Recommended Practice for Software Requirements Specifications*, USA: The Institute of Electrical and Electronics Engineers, 1998.

[13] S. Robertson and J. Robertson, *Mastering the requirements process*, Harlow, UK: Addison Wesley, 1999.

[14] P. Zave and M. Jackson, “Four dark corners of requirements engineering,” *ACM Transactions on Software Engineering and Methodology*, no. *6*, pp. *1-30*, 1997.

[15] M. Odeh and R. Kamm, “Bridging the gap between business models and system models,“ *Information and Software Technology*, no. *45*, pp. *1053-1060*, 2003.

[16] L. Mathiassen, A. Munk-Madsen, P. A. Nielsen and J. Stage, *Objektorienterad analys och design*, Lund, Sweden: Studentlitteratur, 2001.

[17] A. Stellman and J. Greene, *Applied Software Project Management*, O’Reilly, 2006.

[18] K. E. Wiegers, *Software Requirements*, Redmond, USA: Microsoft Press, 1999.

[19] R. C. McIlroy and N. A. Stanton, “Specifying the requirements for requirements specification: the case for Work Domain and Worker Competencies Analyses,” Theoretical Issues in Ergonomics Science, no. 11, 2011.

[20] E. Hull, K. Jackson and J. Dick, *Requirements Engineering*, London, UK: Springer, 2005.

[21] A. Femi, P. Schubert, F. Sudzina and B. Johansson, “Living Requirements Space: An open access tool for enterprise resource planning systems requirements gathering,” *Online Information Review*, no. *34*, pp. *25*, 2010.

[22] J. Nicolás and A. Toval, “On the Generation of Requirements Specifications from Software Engineering Models: a Systematic Literature Review,” *Information and Software Technology*, no. *51*, pp. *1291-1307*, 2009.

[23] U. Eriksson, *Kravhantering för IT-system*, Malmö, Sweden: Studentlitteratur, 2007.

[24] L. Wiktorin, *Systemutveckling på 2000-talet*, Lund, Sweden: Studentlitteratur, 2003.

[25] E. W. Duggan and C. S. Thachenkary, “Higher Quality Requirements: Supporting Joint Application Development with the Nominal Group Technique,” *Information Technology and Management*, no. *4*, pp. *391-408*, 2003.

[26] R. J. Wieringa, *Requirements Engineering: Frameworks for Understanding*, Chichester, John Wiley & Sons Ltd, 1996.

[27] R. A. d. Carvalho, B. Johansson and S. Parthasarathy, “Software tools for requirements management in an ERP system context,” *Journal of Software Engineering and Technology*, no. *2*, pp. *101-106*, 2010.

[28] S. Smith and L. Lai, K. Ridha, “Requirements Analysis for Engineering Computation: A Systematic Approach for Improving Reliability,” *Reliable Computing*, no. *13*, pp. *83-107*, 2007.

[29] IEEE, *(ISO/IEC 12207) Standard for Information Technology - Software Life Cycle Processes*, USA: The Institute of Electrical and Electronics Engineers, 1998.

[30] J. Daniels and T. Bahill, “Requirements and Use Cases,” Systems Engineering, no. *7*, pp. *303-319*, 2004.

[31] M. Thomsen, *Beställarkompetens vid upphandling och utveckling av IT - Om kompetensframväxt i skuggan av kunskapsfragmentering*, in: Department of Informatics, Lund, Sweden: Lund University, 2010.

[32] B. Shackel, “Usability - context, framework, definition, design and evaluation,” in *Human Factors for Informatics Usability*, B. Shackel and S. Richardson, Eds., Cambridge, UK: Cambridge University Press, 1991.

[33] G. M. Olson and J. S. Olson, “Human-Computer Interaction: Psychological aspects of the Human use of Computing,” *Annual Review of Psychology*, no. *54*, pp. *491-516*, 2003.

[34] A. Joshi, N. L. Sarda and S. Tripathi, “Measuring effectiveness of HCI integration in software development processes,” *Journal of Systems and Software*, no. *83*, pp. *2045-2058*, 2010.

[35] P. Kotler and K.L. Keller, *Marketing Management*, Pearson, Prentice-hall, 2009.

Biographical notes

Björn Johansson
Associate professor in information systems at School of Economics and Management, Lund University, Sweden. He received his PhD in Information Systems Development from the Department of Management & Engineering at Linköping University, Sweden in 2007. Previously he worked as a Post Doc at Center for Applied ICT at Copenhagen Business School, involved in the 3rd Generation Enterprise Resource Planning - Strategic Software for Increased Globalization (3gERP.org) research project funded by the Danish National Advanced Technology Foundation. He is a member of the IFIP Working Groups IFIP 8.6 and IFIP 8.9., and the Swedish National Research School Management and IT (MIT).

www.shortbio.net/bjorn.johansson@ics.lu.se

Markus Lahtinen
Lecturer and Doctoral Student in Information Systems at the School of Economics and Management, Lund University, Sweden. Markus holds a M.Sc. in Information Systems since 2001 and a B.Sc. in Business Administration and Economics since 2006. Markus also belongs to the Institute of Economic Research at Lund University, conducting commissioned research for associated industry partners. Since 2006 he has worked with firms like Ericsson, SCA Packaging, Axis Communications, ASSA ABLOY and Niscayah (now part of Stanley Security). His research interest concerns organizational studies and technological change, focusing on the physical security industry.

www.shortbio.net/markus.lahtinen@ics.lu.se

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

A multi-layered software architecture model for building software solutions in an urbanized information system

Sana Bent Aboulkacem Guetat
Le Mans University, ARGUMANS Lab.
Avenue Olivier Messiaen, 72000, Le Mans
France
www.shortbio.net/sana.guetat@univ-lemans.fr

Salem Ben Dhaou Dakhli
Paris-Dauphine University, CERIA Lab.
Place du Maréchal de Lattre de Tassigny, 75575 Paris
France
www.shortbio.net/sdakhli@computer.org

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

A multi-layered software architecture model for building software solutions in an urbanized information system

Sana Bent Aboulkacem Guetat
Le Mans University, ARGUMANS Lab.
Avenue Olivier Messiaen, 72000, Le Mans
France
www.shortbio.net/sana.guetat@univ-lemans.fr

Salem Ben Dhaou Dakhli
Paris-Dauphine University, CERIA Lab.
Place du Maréchal de Lattre de Tassigny, 75575 Paris
France
www.shortbio.net/sdakhli@computer.org

Abstract:
The concept of Information Systems urbanization has been proposed since the late 1990's in order to help organizations building agile information systems. Nevertheless, despite the advantages of this concept, it remains too descriptive and presents many weaknesses. In particular, there is a lack of useful architecture models dedicated to defining software solutions compliant with information systems urbanization principles and rules. Moreover, well-known software architecture models do not provide sufficient resources to address the requirements and constraints of urbanized information systems. In this paper, we draw on the "information city" framework to propose a model of software architecture - called the 5+1 Software Architecture Model - which is compliant with information systems urbanization principles and helps organizations building urbanized software solutions. This framework improves the well-established software architecture models and allows the integration of new architectural paradigms. Furthermore, the proposed model contributes to the implementation of information systems urbanization in several ways. On the one hand, this model devotes a specific layer to applications integration and software reuse. On the other hand, it contributes to the information system agility and scalability due to its conformity to the separation of concerns principle.

Keywords:
information system urbanization; software architecture; software layer; software service; architecture rule; urbanization principle.

DOI: 10.12821/ijispm010102

Manuscript received: 18 January 2013
Manuscript accepted: 4 February 2013

1. Introduction

Almost all modern organizations are faced with more pressures from the ever-changing external economic, technological, social and political environments. Therefore, they have to continuously adapt their priorities, processes products, services and relationships with their partners, customers and suppliers, in order to be compliant with new business rules and market constraints. Moreover, such organizations make today a heavy use of information and communication technologies while implementing their organizational processes. As highlighted by many authors, complexity is among the essential characteristics of organizational information systems [2], [18], [19]. Moreover, Brooks [1], [2] states that information systems have four main properties: complexity, conformity, changeability, and invisibility. In particular, information system complexity is both structural and systemic. The structural complexity of an information system is associated with its structure attributes such as the number and the size of its software applications. The systemic complexity of an information system is related to the interactions between its parts (inter-applications and intra-applications interactions) and the informational flows and services exchanged with external information systems. Information system complexity results in many problems. On the one hand, the difficulty of communication between information system stakeholders leads to poor quality, costs and delays overruns. On the other hand, the difficulty of understanding all the states of software applications, leads to maintenance and evolution problems. Finally, the difficulty of getting a global view of an information system may jeopardize its conceptual integrity. Changeability results in information system evolution. It reflects the need for an organization to continuously adapt its information system in order to take into account its business environment pressures. As stressed by Lehman and Belady [20], information systems applications that are really used change continuously because they are exposed to many forces that require them to change. Information systems evolution is governed by Lehman's law of continuing change which establishes that "*A large program that is used undergoes continuing change or becomes progressively less useful. The change process continues until it is judged more cost-effective to replace the system with a recreated version*" [21].

Software systems aging is another important characteristic of information systems. As stated by Parnas [3], like human aging, software aging is inevitable, but like human aging, there are things that can be done to slow down the process and sometimes even reverse its effects. This author uses the decay metaphor to describe how and why software becomes increasingly brittle over time, and identifies two types of software aging which lead to a decline in the value of a software system: the failure to keep up with changing environment, and the software damages caused by the software changes made. Lehman [21] considers that software aging may be related both to software complexity and software continuous change. According to Lehman's law of increasing complexity, "*as a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it*". Software aging results in decreased performance and reliability due to the software structure deterioration and errors related to changes, and inability to keep up with the market due to increasing size and complexity. In addition to the information systems problems related to the essential characteristics of software, other problems inherent in information systems originate from their accidental characteristics [1]. In particular, many organizations have built their information systems in a chaotic manner materialized by the development and deployment – by each organizational unit – of its own software applications without taking into account redundancies and coherence with applications deployed by other organizational units. Such a way of developing software systems leads to information systems which are complicated, high resource consumers, expensive to maintain, and inflexible. In such a situation, the computerization of any change in organizational processes may be expensive since it mobilizes important resources necessary to identify and modify all the software applications that are affected. According to Perry and Wolf [15], software evolution is strongly dependent on software architecture. These authors describe architecture as the "*load-bearing walls*" of a software system which allows some degree of evolution. In other words, to remain compliant of architectural rules and constraints, software systems architecture allows some changes and precludes others which require a migration to a new architecture. Moreover, some allowed software changes may involve such a migration because they are too expensive to be implemented with the current software architecture. Perry and Wolf [15] consider that software change induces two types of architectural evolution: architectural drift and architectural erosion. The former occurs when software changes are based on a software architecture that is different from the intended architecture. It is due to misunderstanding of the current architecture by the software developers involved in software systems evolution. The latter is caused by violations of software systems architecture and results in increased software systems brittleness. Architectural drift and architectural erosion have negative impacts on software systems maintainability and often lead to architecture redesign.

Information systems problems described in this section impede building agile customer-oriented organizations which need to be supported by open and agile information systems that can be integrated in a secure and efficient mode, with the systems of its customers and suppliers. The concept of information systems urbanization has been proposed since the late 1990's in order to help organizations building agile information systems. Nevertheless, despite the advantages of this concept, it remains too descriptive and presents many weaknesses. In particular, there is a lack of useful architecture models dedicated to defining software solutions compliant with information systems urbanization principles and rules. Furthermore, well-known software architecture models do not provide sufficient resources to address the requirements and constraints of urbanized information systems. In this paper, we draw on the "information city" framework to propose a model of software architecture - called the 5+1 Software Architecture Model - which helps organizations building urbanized software solutions. This framework improves the well-established software architecture models and allows the integration of new architectural paradigms. Our paper is organized as follows. In section 2, we recall the principles of information systems urbanization and present the "information city" framework which constitutes the theoretical foundation of our work. In section 3, we define software architecture and provide a critical analysis of the main software architecture models. Section 4 is dedicated to the presentation of the multi-layered 5+1 software architecture model. In section 5, we conclude this paper by listing its contributions and future research directions.

2. The information city framework

Information systems urbanization is a strategic planning approach for building agile organizational information systems. It includes a set of governance instruments which facilitate the scalability and strengthen the coherence of organizations information systems. This approach is based on four activities: mapping the existing information system, definition of the target information system, gap analysis, and description of the roadmap to reach the target information system. Information systems urbanization is a complex process. The complexity of the information systems urbanization results from the complexity of the information systems artifacts handled by this process. Therefore, to understand information systems urbanization, we use metaphors. Lakoff and Johnson [22] define a metaphor as a way of thinking and seeing that helps understanding one kind of things in terms of another.

The "city planning" or "city landscape" metaphor is among the most popular metaphors proposed in the academic literature to define the foundations of enterprise architecture and information systems urbanization [23], [24], [25]. The application of the "city landscape" metaphor to information systems urbanization has several weaknesses resulting from its descriptive orientation. First, it does not facilitate the identification of architectural principles and rules applicable to complete information systems urbanization. Second, it does not indicate how to manage and take into account the information systems complexity during the construction of urbanized information systems. The "information city" framework [4] generalizes the use of the "city planning" metaphor by stating that - within a modern organization - an information system may be considered as a city where the inhabitants are the applications belonging to this information system. In this city, called the information city, the common parts are software artifacts and information shared by all the information system applications while the private parts are composed of software artifacts and information owned by each application. An application belonging to the information city behaves as a master of its proper data and artifacts and as a slave regarding data and software artifacts which belong to other applications. That means that an application can use, update or suppress data and artifacts it owns but can only use a copy of other applications data and software artifacts.

Comparing an information system to a city extends the use of the "city landscape" beyond the analogy between software and building construction by emphasizing the problem of information system governance. On the one hand, following the example of a city, the relationships between the applications which populate the information city must be managed. That means that a set of architecture principles and rules has to be specified in order to govern exchanges either between application belonging to an information system or between such applications and the external environment like other information systems or end-users. On the other hand, the vast number of application assets in combination with the natural expansion of the application portfolio, as well as the increasing complexity of the overall information system, drives a need for the information system governance. Therefore, the “information city” framework permit defining architecture principles and rules which help organizations prioritize, manage, and measure their information systems.

Using the “information city” framework makes organizations able to apply a structure for classifying information system applications, functions, or services in a coherent way. It defines responsibility plots from coarse to fine-grained into discrete areas, which together form the complete Information City Plan (ICP) (Fig. 1).

The ICP is a set of areas, districts, and blocks. An area is composed of districts and a district splits into blocks. The ICP areas are determined according to three urbanization principles resulting from a deep analysis of the organization’s and information technology strategies. These principles are:

- Urbanization principle 1: Determine front-office *vs.* back-office responsibilities;
- Urbanization principle 2: Specialize front-office and back-office regarding the organization’s processes;
- Urbanization principle 3: Identify the components common to the back-office and the front-office**.**

The first architecture principle - Determine front-office *vs.* back-office responsibilities - identifies the responsibilities of the organization’s front-office and back-office. The front-office is dedicated to management of the relationships with the organization’s external environment while the back-office is dedicated to the development of products and services. For instance, within an insurance company the back-office manages the insurance and services commitments whatever the distribution channels.

To apply the second architecture principle - specialize front-office and back-office regarding the organization’s processes - we use a classification of organizational processes into five categories: business processes, support processes, decision-making processes, communication with the organization’s external environment processes, and management of the relationships with the organization’s external environment processes. The first three categories relate to organizational processes in the back-office, while the last two refer to those in the front-office. According to this classification, the second architecture principle permits identifying at least three areas in the organization’s back-office and two areas in the organization’s front-office. The back-office areas are:

- The “Business Intelligence area” associated with decision-making processes;
- The “Support area” associated with support processes;
- And at least one “Business area” associated with business processes: the “Policy and Claims area” is an example of a business area within an insurance company.

The front-office areas are the “Inbound and Outbound flows Management area” and the “Party Relationships area”. The “Inbound and Outbound flows Management area” is associated with the communication with the organization’s external environment processes. This area is dedicated to the management of the informational flows exchanged by an organization and its external environment. It describes the various technology channels used by an organization while exchanging information with its external environment. The “Party Relationship area” is associated with management of the relationships with the organization’s external environment processes. This area supports the relationships linking an organization with its customers and partners whatever the communication channel.

The third architecture principle - Identify the components common to the front-office and the back office - refers to either the components that link the front-office and the back-office or the artifacts shared by the back-office and the front-office. The application of this principle results in identifying two areas: an “Integration area” and a “Shared information area”. The first area allows exchanges of informational flows and services between the back-office and the front-office applications.

The second area contains information shared by all the applications of the organization's information system as well as the applications which manage shared information data. The customers and products repositories are examples of information shared by all the applications of an organization's information system.

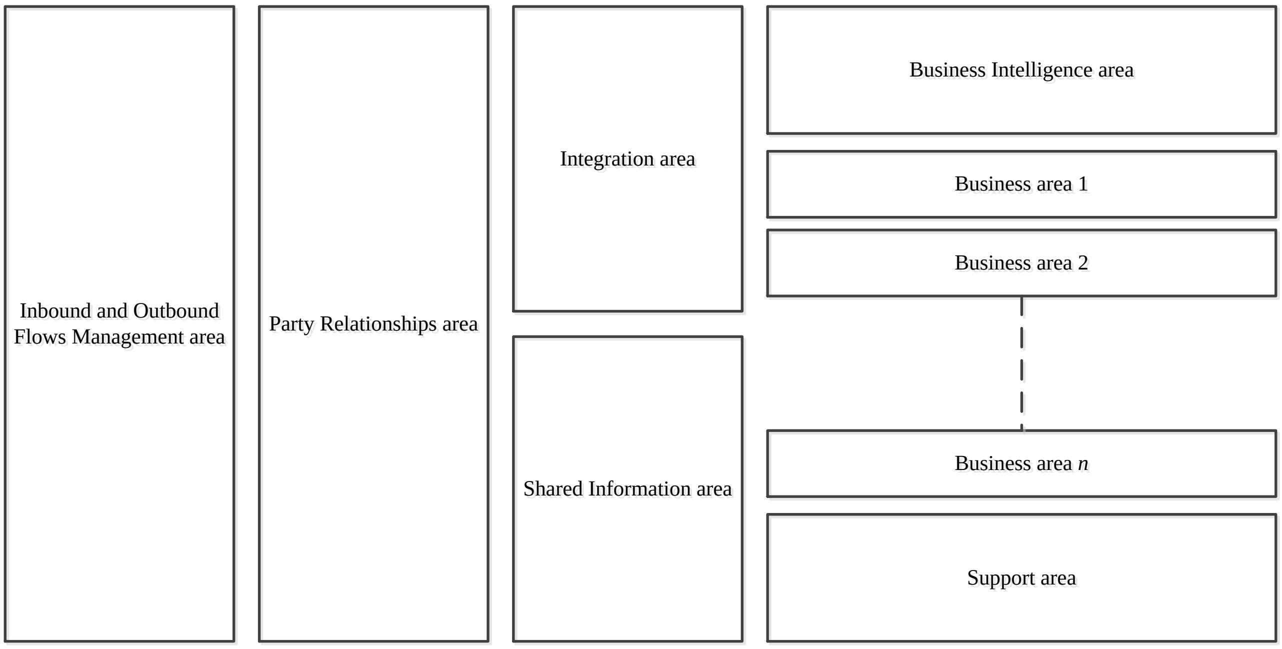

Fig. 1. The Information City Plan (ICP)

3. Software architecture: definitions and critical analysis

As stressed by many authors, software architecture has emerged as an important field of information systems for managing software applications development, evolution, and maintenance [7]-[10]. The main intent of software architecture is to provide intellectual control over a complex software system [11]. Indeed, software architecture models the structure and behavior of a system; and presents a high level view of a system, including the software elements and the relationships between them. Software architecture is a complex concept that is difficult to capture in a single definition. Many definitions of the software architecture concept are proposed in the literature. For example, Toffolon [5] and Toffolon and Dakhli [6] consider that software architecture describes a software solution which computerizes an organizational solution of an organizational problem. Garlan and Shaw [26] stress that software architecture is characterized by a set of issues which include gross organization, global control, structure, communication protocols, and assignment of functionality to design elements. Kruchten [12] and Jansen [27] draw on work by Shaw and Garlan [10] to define software architecture as the set of significant decisions about the organization of a software system. Such decisions focus on the selection of the structural elements and their interfaces by which a system is composed, the behavior as specified in collaborations among those elements, the composition of these structural and behavioral elements into larger subsystem, and architectural style that guides this organization. These authors point out that software architecture also involves usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns. Bass et al. [7] propose a definition of software architecture that acknowledges that architecture of a single software system may be described using different types of structures. According to these authors, software architecture is based on the structure or structures of a software system, which includes software elements, the externally visible properties of those elements, and the relationships among them. The IEEE 1471-2000 standard [13] defines software architecture as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. In addition to the software architecture components and their relationships, this definition covers architecture rules and principles like architectural styles or the use of particular conventions during the software development, maintenance, and evolution life cycles.

Due to the complexity of the software architecture concept, two problems have to be solved in order to understand it and apply it efficiently in software development, maintenance, and evolution projects: software architecture description, and software architecture modeling. The first problem means that it is difficult to describe software architecture by one structure, or by one type of abstraction. To deal with this problem, academics and practitioners suggest using views and perspectives. To this end, Kruchten [28], Hofmeister et al. [29], and Clements et al. [30] stress that multiple views are required to completely describe and document software architecture. Each view of a software architecture addresses a specific set of concerns and is created using guidelines defined in a viewpoint.

According to Jansen et al. [14], software architecture has three perspectives: blueprint, roadmap, communication vehicle, and quality predictor. The blueprint perspective outlines software systems structures and behaviors. The roadmap perspective describes the evolution paths of software systems. The communication vehicle perspective focuses on the communication instruments used by the software system stakeholders to share the architectural decisions in order to steer and influence the design of a software system. The quality predictor perspective provides an early predictor of the quality of software systems architectures. As noted above, software architecture is inevitably subject to evolution due to software aging due to architectural drift and architectural erosion [15].

To solve the second problem and support information systems evolution without compromising their invariants and integrity, many software architecture models have been proposed by academics and practitioners such as the Perry and Wolf's architecture model [15], the product line model [31], and the multi-layered architecture like the Model-View-Controller (MVC) [33], the Presentation-Abstraction-Control (PAC) [16], and the multi-tiers architecture models. In particular, the architecture model proposed by Perry and Wolf [15] consists of design elements, form, and rationale. Firstly, design elements include data, processing, and connecting elements. Secondly, forms refer to the relationships among the elements of software architecture. Finally, rationale describes the motivation for the decisions that yield a particular set of elements and form.

The software architecture product-line model has been proposed by Clements and Northrop [31] to increase software reuse through the use of architectural rules and principles. A software product-line encompasses a whole range of software artifacts that have common characteristics. It involves the development of product-line assets, such as a product-line architecture, reusable components, and product-line members. The assets that apply to the product line as a whole are developed in a process referred to as domain engineering while the product-line members are developed in a process called application engineering. According to [31], the development of such products as a software product-line makes their commonalities and variability explicit in a product-line architecture. Therefore, the development of individual products consists in binding the variation points defined in the product-line architecture to specific instances [32].

The multi-tiers architecture model is a multi-layered software architecture model which provides a logical way to separate the different responsibilities of software applications. For example, according to the three-tier software architecture model, a software system is composed of three parts called tiers:

- The presentation tier is responsible for displaying information and supporting the interactions with the end-users;
- The application tier - also called business tier - is responsible for the coordination of the software system business logic, *i.e.*, it executes commands, actions and moves data between the presentation and the data tier;
- The data tier - also called persistence tier - is responsible of data retrieving and storage.

The software system tiers should be as independent as possible from each other. The MVC model suggests that a software system is based on three components: the Model component, the View component, and the Controller component [32]-[33]. The Model component provides the core functionality of the software system. The View component role consists in presenting models from the Model component. The Controller component manages interactions between end-users and views from the View component, and checks how views and models are manipulated by end-users. A model may be associated with many views which are notified by this model whenever it is updated. This enables developers to create or modify views without altering the associated model, and guarantees that all views associated with a model are synchronous since they reflect the same model state. Although the MVC model is different from the multi-tiers architecture model, we think that the View and Controller components belong to the Presentation tier while the Model component belongs to the application and data tiers. The same can be applied to the PAC model.

Through the separation of responsibilities, the multi-layered software architecture model facilitates the management of many aspects of the information systems complexity, and improves software systems maintenance and evolution. Nevertheless, many important problems remain unsolved. For example, these models do not provide efficient ways for cooperation either within the same information system or between many information systems. Moreover, the existing multi-layered software architecture models do not consider the constraints and rules of urbanized information systems. In other words, the multi-layered models (multi-tiers, MVC, PAC, etc.) need to be enhanced in order to manage efficiently the complexity inherent in multi-channels access and navigation, or services and information flow exchanges.

4. The multi-layered 5+1 software architecture model

An urbanized application is a software system whose architecture is compliant with the information city goals such as agility and reuse. This means that an urbanized application must meet the basic architectural rules and principles induced by urbanization constraints. “Strong coherence and weak coupling”, “separation of concerns”, “standard communication protocols”, and “data hiding” are examples of such principles. Moreover, an urbanized application should take into account the four urbanization principles used to build the Information City Plan (ICP). As a result, an urbanized application is organized as a set of parts that have public resources and private resources, and interact by using standard communication protocols. Therefore, the architecture of an urbanized application is organized in layers, each layer being responsible for a specific concern. In this section, we propose a software architecture model - called the 5+1 model - to help design urbanized applications.

The multi-layered 5+1 model, which describes the architecture of software systems belonging to urbanized information systems, is composed of six architecture layers: the Interface layer, the Navigation layer, the Orchestration and Choreography layer, the Services layer, the Data Access layer, and the Technical Services layer. Each layer is associated with a data set which describes its modifiable parameters. These parameters are stored in a read-only repository - called layer repository - which enables their update without modification of the software system programs. Fig. 2 illustrates the multi-layered 5+1 software architecture model.

4.1 Presentation of the 5+1 model layers

The Technical Services layer includes technical services shared by the other layers. Security services, network services, errors management services, and middleware services are examples of technical services managed by this layer. The detailed description of this layer is outside the scope of this paper. Table 1 provides a synthetic description of the other five layers of the 5+1 software architecture model which includes information related to the role of each layer, the functions it supports, the content of its repository, and the associated architecture rules.

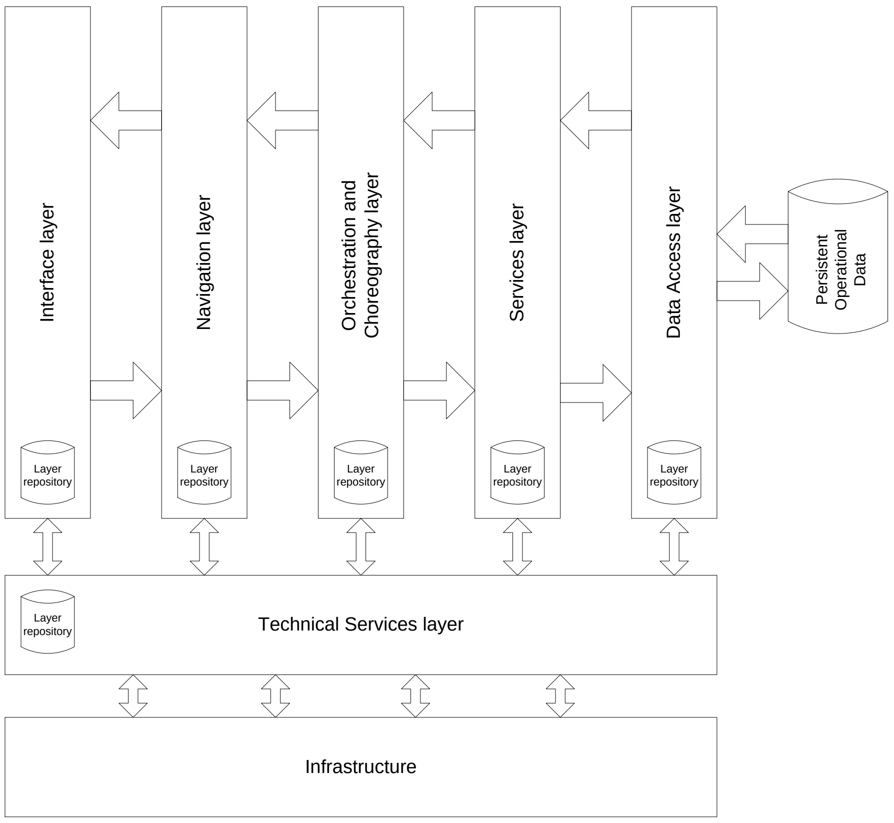

Fig. 2. The multi-layered 5+1 software architecture model

Table 1. The description of the 5+1 model layers

Layer	Role	Supported functions	Layer repository (Examples	Architecture rules (Examples
Interface	- Management the interactions with end-users for each technical communication channel - Management of the graphical aspects of the human-machine interface - Components: screens, editions, and formatting elements	-Presentation management - Screens content display - Syntax control of data - Surface control of input data - Online help	- Displaying colors - Messages labels - Screens associated with a language	- **Rule 1:** Only the modules of the Interface layer can interact with human end-users
Navigation	- Description of the progress of the screens kinematics - Management of data specific to the interaction between the software system and its human end-users - Management of a context related to informational flows exchanged with the Orchestration and Choreography layer - Components: technical communication channels kinematics (Internet, Mainframe 3270, …)	- Identification of the screens for tasks performing - Calls to the Orchestration and Choreography layer modules to carry out controls related to the organizational processes supported by the software system - Routing of displayed information to local printers (forms, display styles, …)	- List of screens associated with a task	- **Rule 1:** Only the modules of the Navigation layer can call the Orchestration layer modules

Table 1. The description of the 5+1 model layers (cont.)

Layer	**Role**	**Supported functions**	**Layer repository (Examples**	**Architecture rules (Examples**
Orchestration and Choreography	- Identification of the organizational processes activities supported by the software system - Management of the sequence of tasks supported by the software system - Description of the end-users roles - Control of the informational and services exchanges with other software systems - Management of a context related to tasks running in order to allow interruptions without data publication	- Start and complete a sequence of tasks - Expose services to other software systems - Call of services exposed by other software systems - Sequence of services invocation - Services orchestration - Services choreography	- List of tasks making up a use case - List of services used by a use case - Description of the sequence of tasks making up a use case	- **Rule 1:** The business data processed during process execution are managed in a process context. This context is managed exclusively by a module of the Orchestration and Choreography layer. No module of another layer can access it even in reading-only - **Rule 2:** Only the modules of the Orchestration and Choreography layer can exchange informational flows and services with other software systems - **Rule 3:** Only the Orchestration and Choreography layer can expose services for other software systems
Services	- Hosting the rules applicable to the software system informational entities - Implementation of the functions processing the software system informational entities	- Description of the state of the software system informational entities - Carrying out simple and complex controls - Data processing - Description of the services supported by the software system - Recording of status changes of the software system informational entities	- Computing rules - Information related to data processing: interest rates, legal information, …	- **Rule 1:** The Services layer guarantees the inter-business process consistency through checking the software system informational entities
Data Access	- Providing access to operational persistent data belonging to the software system by ensuring a real independence between processing and physical data models and performing data integrity controls	- Data selection - Data update - Data deleting - Data creation - Data edition - Table joining - Data integrity control	- List of tables to be used to store information related to an informational entity	- **Rule 1:** Only the modules of the Data Access layer provide data read and write services of operational persistent data belonging to the software system - **Rule 2:** Read and write services exposed by the Data Access layer are defined according to logical data model of the software system

4.2 Complementary information on the Orchestration and Choreography layer

We note that orchestration and choreography describe two complementary concepts related to processes execution. Orchestration describes how a central entity - called the coordinator - manages dependencies during the execution of services involved in a higher-level organizational process. Choreography focuses on the interactions between collaborating entities which may have their own internal orchestration processes. Such interactions are based on protocols that enable the conversation between the parties involved in choreography. According to [17], in choreography no organization necessarily controls the collaboration logic while orchestration is generally owned and operated by a single organization.

Orchestration and choreography are part of the organizational processes control, distributed across the ICP Integration area (Internal and external control applications) and the other information system applications whose orchestration and choreography layer is responsible for use cases monitoring. Each organizational process has a context containing temporary information including its status, newly created or modified information, and information already read. The process context is composed of two nested sub-contexts related to the information system and application levels. At the information system level, organizational processes contexts are managed by an application - called the Business Processes Management System (BPMS) - which belongs to the ICP Integration area, and updated based on information recorded by applications at the end of use cases. At the application level, the Orchestration and Choreography layer manages the sequence of tasks supported by the software system and the contexts of its use cases. Each use case is associated with a component - called use case driver - in the Orchestration and Choreography layer. Only use case drivers are allowed to access to use cases contexts. To implement the Orchestration and Choreography layer, a good practice consists in distinguishing two types of services: the transition services, and the request services. The former updates the use cases contexts while the latter’s role is limited to reading these contexts. Apart from information already read and managed by the application, the content of the process context consists of information generated during the execution of this process, or collected from other applications. Therefore, information manipulated by a service managed in the 5+1 model Services layer are either persistent operational data accessed through the Data Access layer or temporary data provided by the context of the process with calls this service.

In addition to the management and internal control of the interactions between processes supported by an application, system, the Orchestration and Choreography layer manages the application interactions with other information system applications. Therefore, the layer is composed of three parts: Processes Internal Control (PIC), Management of Inbound Informational Flows (MIF), and Management of Outbound Informational Flows (MOF). Within the Orchestration and Choreography layer, the steering and control role is devoted to the PIC part while the MIF and MOF parts are informational flows converters and thus play a role similar to the Interface layer role.

4.3 Management of services in the 5+1 software architecture model

The 5+1 software architecture model manages software services according to many perspectives. Firstly, a service is either public or private. Secondly, a service is exposed by a software system either for end-users or for other software systems. Thirdly, a service is either used only by the information system applications or may be used by external information systems like partner’s information systems. Finally, services may be viewed as integrating means of information systems. Therefore, the management of services in the 5+1 software architecture model is based on a typology which distinguishes five types of services: information system service, applicative service, end-user service, layer service, and component service. An information system service is a software service exposed by an information system for external information systems, and accessed via a specific application belonging to the Inbound and Outbound Flow Management Area of the Information City Plan (ICP). An applicative service is a software service exposed by an application for other applications belonging to the same information system, and accessed via the Orchestration and Choreography layer. An end-user service is a software service exposed by an application for human end-users, and accessed via the interface layer. Information system services and end-user services are usually composed of several applicative services. A layer service is a software service exposed by a layer for the other layers of a software system. A component service is a software service exposed by a component for the components of the same software system layer. Table 2 explains when a service is public and when it is private.

4.4 The relationships between the 5+1 software model and the ICP areas

The software layers identified by the 5+1 software architecture model conform to the separation of concerns principle. Therefore software systems architected according to this model are scalable and adaptable to user needs. As a result, the importance of the layers of such applications depending on the ICP area hosting them can be taken into account during the development life cycle.

Table 2. Typology of services

Service	**Public for**	**Private for**
Information System service	- External Information Systems	- Information System applications - Human end-users
Applicative service	- Applications belonging to the same Information System	- External Information Systems - Human end-users
End-user service	- Human end-users	- Information System applications - External Information Systems
Layer service	- Other layers of the same software system	- Information System applications - External Information Systems - Human end-users
Component service	- Other components of the same layer	- Information System applications - External Information Systems - Human end-users - Other layers of the same software system

Table 3 provides an assessment of the importance of the layers of a software system implemented in a French insurance company and architected according to the 5+1 model. This company operates in many countries all over the world with more than 30,000 employees. It covers all insurance sectors including mass-risks such as motor car liability insurance or accident insurance, and industrial policies required by international companies. In recent years, it has reinforced its business in the personal insurance sector, income capacity and savings management, in particular through the promotion of pension-based life products. Lately, it has expanded its business from insurance to the wide area of asset management and financial services, and established a full-fledged bank structure in order to optimize the products and services. The current information system of this company is composed of more than a thousand applications running either on mainframes or on open systems. An urbanization project of this information system is ongoing to reach a target urbanized information system consistent with the ICP and the applicable architecture rules and principles. To assess the importance of the 5+1 software architecture model layers with respect to the ICP areas to which applications belong, we studied the architectures of many urbanized applications implemented according to the 5+1 software architecture model. We also collected additional information through interviews with the information system architects involved in the design and implementation of these applications. To complete Table 3, we have used a five-point Likert scale (0=Not important at all, 1=Weakly important, 2=Moderately important, 3=Important, 4=Very important). We note that the Technical Services layer is not included in Table 3 since it is important regardless of the ICP area. This table provides several indications. For example, it shows that the Interface and Navigation layers are very important for applications hosted by the Inbound and Outbound Flow Management Area while they are not relevant for applications belonging to the Shared Information area. Moreover, the Data Access layer is very important for applications hosted by the Business Intelligence area, the Shared information area, and the Business area.

Table 3. Importance of software layers depending on ICP areas

Areas	Software layers				
	Interface	**Navigation**	**Orchestration and Choreography**	**Services**	**Data Access**
Inbound and Outbound Flow Management	4	4	1	1	0
Party Relationships	1	1	3	3	3
Business Intelligence	3	0	1	1	4
Integration	1	1	4	1	1
Shared Information	0	0	2	3	4
Support	2	1	3	2	3
Business	1	1	3	4	4

Table 3 also shows that the most important layers are:

- Interface and Navigation for Inbound and Outbound Flow Management area applications;
- Orchestration and Choreography, Services, and Data Access for Party Relationships area applications;
- Interface and Data Access for Business Intelligence area applications;
- Orchestration and Choreography for Integration area applications;
- Services and Data Access for Shared Information area applications;
- Orchestration and Choreography and Data Access for Support area applications;
- Orchestration and Choreography, Services, and Data Access for Business area applications.

5. Conclusion and future research directions

In this paper, we have presented a software architecture model - called the 5+1 model - which helps build urbanized software systems. This model is compliant with information systems urbanization principles. First of all, the 5+1 model is organized in the same way than the Information City Plan (ICP) since it addresses the main urbanization principles used to define the ICP. On the one hand, the first urbanization principle is addressed by the 5+1 model which permits identifying - for each software system - a front-office composed of the Interface and the Navigation layers, and a back-office composed of the Orchestration, Services, and Data Access layers. On the other hand, the second urbanization principle is reflected by the 5+1 model which distinguishes four main processes supported by the back-office layers and two main processes supported by the front-office layers. Processes supported by the back-office layers are: Communication with the Information System applications, Tasks Management, Services Management, and Data Access Management. The Communication with the Information System applications and the Task Management processes are supported by the Orchestration and Choreography layer. The Services Management process and the Data Access processes are respectively supported by the Services and the Data Access layers. Processes supported by the front-office layers are: Management of the presentation of information and static aspects of interactions with end-users for each technical communication channel, and Management of the progress of screens kinematics. The former is supported by the Interface layer while the latter is supported by the Navigation layer. Moreover, the third urbanization principle is taken into account by the 5+1 model since the Technical services layer is shared by the front-office and the back-office while the Orchestration and Choreography layer allows front-office and back office to communicate.

Secondly, the 5+1 software architecture model contributes to the implementation of information systems urbanization in several ways. On the one hand, this model devotes a specific layer - the Orchestration and Choreography layer - to applications integration and software reuse. Indeed, software services exposed by information system applications for reuse facilitate the integration of the applications using them. On the other hand, the 5+1 model contributes to the information system agility. As stressed previously, the 5+1 model conform to the separation of concerns principle. Therefore, a software system architected according to this architecture model is scalable and adaptable to users' needs. In particular, the importance of the layers of such applications depending on the ICP area hosting them can be taken into account during the development life cycle. As a result, computerization resources can be allocated efficiently according to the importance of the layers of the software system under development. However, this model should be evaluated through experimentation in order to better use it in practice. Furthermore, two questions remain unanswered. The first question concerns the effectiveness of using the 5+1 model for architecting a Decision Support Systems and the second is related to the integration of software systems architected using this model with enterprise systems like ERP and CRM systems. These issues are two future research directions.

References

[1] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, vol. *20*, no. *4*, pp. *10-19*, April, 1987.

[2] F. P. Brooks, Jr., *The Mythical Man-Month*, Boston, MA, USA: Addison-Wesley Longman Inc., 1995.

[3] D. L. Parnas, "Software Aging," in *Proceedings of the 16th International Conference on Software Engineering* (*ICSE'94*), IEEE Computer Society/ACM Press, Sorrento, Italy, May 16-21, 1994, pp. *279-287*.

[4] S. Guetat and S. B. D. Dakhli, "The Information City: A Framework for Information Systems Governance," in *Proceedings of the 5th Mediterranean Conference on Information Systems* (*MCIS'2009*), AIS Digital Library, 2009.

[5] C. Toffolon, "L'Incidence du Prototypage dans une Démarche d'Informatisation," PhD. Dissertation, Université de Paris-IX Dauphine, Paris, France, 1996.

[6] C. Toffolon and S. Dakhli, "The Software Engineering Global Model," in *Proceedings of the COMPSAC'2002 Conference*, IEEE Computer Society Press, Oxford, United Kingdom, August 26-28, 2002, pp. *47-53*.

[7] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*, 2nd edition, Reading, MA, USA: Addison-Wesley, 2003.

[8] G. Booch, "The irrelevance of architecture," *IEEE Software*, vol. *24*, no. *3*, pp. *10-11*, 2007.

[9] R. N. Taylor, N. Medvidovic and E. Dashofy, Software *Architecture: Foundations, Theory, and Practice*, New York, NY, USA: Wiley Publishing, 2009.

[10] M. Shaw and D. Garlan, *Software Architecture: Perspectives on a Emerging Discipline*, Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[11] P. Kruchten, H. Obbink and J. Stafford, "The past, present and future for software architecture," *IEEE Software*, vol. *23*, no. *2*, pp. *2-10*, 2006.

[12] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd edition, Reading, MA, USA: Addison-Wesley Professional, 2003.

[13] Software Engineering Standards Committee of the IEEE Computer Society. (2000, September, 21). *IEEE Recommended practice for architecture description of software-intensive systems, IEEE Std 1471-2000,* IEEE-SA Standards Board [Online]. Available: http://standards.ieee.org/

[14] A. Jansen, J. Bosch and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *Journal of Systems and Software*, vol. *81*, no. *4*, pp. *536-557*, 2008.

[15] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture. Software Engineering Notes," *ACM SIGSOFT,* vol. *17,* no. *4,* pp. *40-42*, 1992.

[16] J. Coutaz, "Architectural design for user interfaces," in *Encyclopaedia of Software Engineering,* J. J. Marciniak, Ed., New York, NY, USA: John Wiley & Sons, 1994, pp. *38-49*.

[17] T. Erl, *Service-oriented architecture: concepts, technology, and design*, Upper Saddle River, NJ, USA: Prentice-Hall International, 2005.

[18] C. Toffolon, "The Software Dimensions Theory," in *Enterprise Information Systems, Selected Papers Book*, Joaquim Filipe, Ed., Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[19] P. P. Lemberger, *Managing the Complexity of Information Systems: The Value of Simplicity*, New York, NY: Wiley & Sons, 2011.

[20] M. M. Lehman and L. A. Belady, Eds., *Program evolution: processes of software change,* Waltham, MA, USA: Academic Press, 1985.

[21] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *IEEE Proceedings*, vol. *68*, no. *9*, pp. *1060-1076,* 1980.

[22] G. Lakoff and M. Johnson, *Metaphors we live by,* 2nd Ed. Chicago, IL, USA: University of Chicago Press, 2003.

[23] D. Dieberger and U. F. Andrew, "A City Metaphor to Support Navigation in Complex Information Spaces," *Journal of Visual Languages and Computing*, vol. *9*, no. *6*, pp. *597-622*, 1998.

[24] C. Knight, "System and Software Visualization," in *Handbook of software engineering & knowledge engineering. (vol. 2): Emerging technologies*, Change S. K., Ed., River Edge, NJ, USA: World Scientific Publishing Company, 2002, pp. *131-139*.

[25] J. Ross, "Creating a strategic IT architecture competency: Learning in stages," *MISQ Quarterly Executive*, vol. *2*, no. *1*, pp. *31-43*, 2003.

[26] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering*, vol. *2*, pp. *1–39*, Hackensack, NJ, USA: World Scientific Publishing Company, 1993.

[27] A. Jansen, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, IEEE Computer Society Press, Pittsburgh, USA, 2005.

[28] P. B. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. *12*, no. *6*, pp. *42-50,* 1995.

[29] C. Hofmeister, R. Nord and D. Soni, *Applied Software Architecture*, Reading, MA, USA: Addison-Wesley Publishing, 1999.

[30] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, *Documenting Software Architectures:Views and Beyond*, Reading, MA, USA: Addison-Wesley Publishing, 2002.

[31] P. Clements and L. Northrop, *Software Product Lines, Practices and Patterns*, Reading, MA, USA: Addison-Wesley Publishing, 2002.

[32] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*, Reading, MA, USA: Addison-Wesley Publishing, 2000.

[33] A. Hussey and D. Carrington, “Comparing the MVC and PAC architectures: a formal perspective,” *IEEE Proceedings on Software Engineering*, vol. *144*, no. *4*, pp. *224-236*, 1997.

[34] G. E. Krasner and S. T. Pope, “A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80,” *JOOP*, vol. *1*, no. *3*, pp. *26-49*, 1988.

Biographical notes

Sana Guetat
Sana Guetat received the Master’s degree in Accounting from the University of Manouba, Tunisia and the PhD. degree in Management Science from the Toulouse I University, Toulouse, France.
She is currently an Assistant Professor of Management Science, and Information Systems at the Maine University, Le Mans, France, where she is a member of the ARGUMANS Laboratory and the head of the Master of Business Administration. Her research interests include Accounting, Finance, Audit, Information Systems Architecture, and Knowledge Management.

www.shortbio.net/sana.guetat@univ-lemans.fr

Salem Ben Dhaou Dakhli
Salem Ben Dhaou Dakhli received a Statistician Economist Diploma from the National School of Statistics and Economic Administration (ENSAE), Paris, France, the Master’s degree in Applied Mathematics, the Master’s degree in Computer Science, and the PhD. degree in Computer Science from the Paris-Dauphine University, Paris, France. He is currently a Professor of Statistics, decision theory, software engineering, and Information Systems Architecture and a member of the CERIA Laboratory at the Paris-Dauphine University. His research interests include Software Engineering, Information Systems Architecture, Software Economics, and Knowledge Management. Dr. Dakhli is a TOGAF Enterprise Architect certified by the Open Group and a consultant in Information Systems with many French Banks and Insurance Companies.

www.shortbio.net/sdakhli@computer.org

SciKA

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

A case study on variability management in software product lines: identifying why real-life projects fail

Tom Huysegoms
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven
Belgium
www.shortbio.net/tom.huysegoms@kuleuven.be

Monique Snoeck
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven
Belgium
www.shortbio.net/monique.snoeck@kuleuven.be

Guido Dedene
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven
Belgium
www.shortbio.net/guido.dedene@kuleuven.be

Antoon Goderis
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels
Belgium
www.shortbio.net/antoon.goderis@kuleuven.be

Frank Stumpe
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels
Belgium
www.shortbio.net/frank.stumpe@kuleuven.be

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

A case study on variability management in software product lines: identifying why real-life projects fail

Tom Huysegoms
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/tom.huysegoms@kuleuven.be

Monique Snoeck
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/monique.snoeck@kuleuven.be

Guido Dedene
Faculty of Business and Economics, Management Information Systems Group, KULeuven
Naamsestraat 69, 3000 Leuven, Belgium
www.shortbio.net/guido.dedene@kuleuven.be

Antoon Goderis
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels, Belgium
www.shortbio.net/antoon.goderis@kuleuven.be

Frank Stumpe
KBC ICT, KBC Global Services
Havenlaan 2, 1080 Brussels, Belgium
www.shortbio.net/frank.stumpe@kuleuven.be

Abstract:
Economies of scale can be seen as some kind of “holy grail” in state of the art literature on the development of sets of related software systems. Software product line methods are often mentioned in this context, due to the variability management aspects they propose, in order to deal with sets of related software systems. They realize the sought-after reusability. Both variability management and software product lines already have a strong presence in theoretical research, but in real-life software product line projects trying to obtain economies of scale still tend to fall short of target. The objective of this paper is to study this gap between theory and reality through a case study in order to see why such gap exists, and to find a way to bridge this gap. Through analysis of the causes of failure identified by the stakeholders in the case study, the underlying problem, which is found to be located in the requirements engineering phase, is crystallized. The identification of a framework describing the problems will provide practitioners with a better focus for future endeavors in the field of software product lines, so that economies of scale can be achieved.

Keywords:
software product lines; variability management; harmonization & variabilization; requirements engineering; case study.

DOI: 10.12821/ijispm010103

Manuscript received: 28 February 2013
Manuscript accepted: 18 March 2013

1. Introduction

Variability between related software systems offers a challenge in an otherwise dull world where each software system would be the same as the previous one. At the same time, such variability can quickly turn into a nightmare for those who need to develop or maintain a set of related software systems. While in theoretical research terms like variability management and software product lines are commonplace, in practice there are still a lot of companies that struggle with variability and with the problems variability brings about. Previous research [1] through an exploratory study in a software intensive company revealed that variability did induce difficulties in practice, so an interesting problem in the domain of software engineering was identified. How is it possible that companies still struggle with variability while in the academic research field there is an abundance of theoretical studies about the problems that can be encountered in practice, and more important, how can companies be offered a way to overcome their problems? The research question that is addressed in this research is the following: "What are possible reasons behind the failures in practice of projects that develop sets of related software systems in order to obtain economies of scale, and how can these problems be resolved?"

Using a case study as the start of our research ensures to a certain extent that the results obtained are usable in real companies. In the rest of this paper, findings on a real life case conducted in a large scale bank and insurance company are analyzed and reported. The second section positions the research against related work. Section 3 discusses the methodological approach opted for. Section 4 describes the context of the empirical case study. The fifth section identifies the different kinds of stakeholders and presents the results of the case study. These results serve as starting point to crystallize the problems impeding successful variability management and achieving economies of scale in practice. Section 6 provides the description of a framework representing the variability decisions that need to be taken during the phase of requirements engineering, which is the phase where the problem is situated according to the case study results. The seventh and last section recaps the research questions addressed and draws the conclusions.

2. Related work

In the past decades substantial research effort has been devoted to solving the issues concerning variability in related software systems. Out of this effort multiple ways to manage variability have been developed. As early as in 1990 variability management was already identified by Kang et al. [2] as part of Feature-Oriented Domain Analysis (FODA). Variability management is defined by Schmid and John [3] as encompassing the activities of explicitly representing variability in software artifacts throughout the lifecycle, managing dependencies among different variabilities, and supporting the instantiation of the variabilities. The need for a clear understanding of variability in the context of requirements engineering is quite obvious, as not only research on the theoretical concept of variability itself is done [4], research on related subjects like e.g. coping with preferences in requirements [5] suggests the same need for a clear understanding of variability. Following the seminal work of FODA, multiple variability management approaches have been developed. Some of these were extensions of the original FODA specification like e.g. FORM [6]. Other approaches were developed more independently like KobrA [7] and COVAMOF [8]. More recent variability management approaches that have received considerable attention can be to some extent retraced to this core strand of research. The Orthogonal Variability Model (OVM) approach by Pohl et al. [9] provides a good example of this. For a more complete overview of the variability management literature we refer the reader to Chen et al. [10], which revises the field of variability management. In summary, it can be stated that the variety and plethora of available approaches indicate that significant attention has been given to the issue by the research community.

The notion of software product lines, is closely related to the issues considered in the strand of research described in the previous paragraph. Software product lines have been introduced in literature at about the same time as variability management has been inducted in research. The much referenced work of Clements et al. [11] defines a software product line as follows: "*A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a* *common set of core assets in a prescribed way*". The Carnegie Mellon Software Engineering Institute defines it as follows on its website [12]: "*A software product line epitomizes strategic, planned reuse. More than a new technology, it is a new way of doing business. Organizations developing a portfolio of products as a software product line are experiencing order-of-magnitude improvements in cost, time to market, and productivity*". Both quotations together stress the major things about a set of related software systems that need to be considered. A software product line is a portfolio or set of relating software systems, also called a product family, which is developed with a mission to improve the way to do business. It therefore affects business and goes beyond its Information and Communication Technology (ICT) context. A software product line has a focus on a specific market segment, in which the software systems are closely related so economies of scale of all sort can be obtained. It can thus be said that software product lines are the sets of software systems that are developed in order to obtain economies of scales.

The most important thing in a software product line is the variability amongst the members of the set, as this is the contrasting factor between the instances of the software product line. The management of this variability is key to make the software product line a success. If the basis, on which the instances are built, the core asset, offers the proper amount of customization through variability, then the product line can reap the benefits attributed to it in literature. The importance of variability in software product lines also explains the relatedness of software product lines research with variability management research.

3. Research methodology

In the domain of software engineering, it was acknowledged by Moody et al. [13] that there exists a problem of knowledge transfer between theory and practice, also known as the technology transfer problem. Kaindl et al. [14] argued that such gap also exists in the domain of requirements engineering, the phase that should primarily deal with variability problems. These studies propose a number of approaches to bridge this technology transfer gap. The objective of all these approaches is to bring theoretical research closer to real life by providing a knowledge transfer channel. The research in this paper starts from a real life case in order to develop and explain theoretical knowledge. By doing so the knowledge transfer channel is present right from the start of the research. Through case study research [15], an empirical research technique, it is largely ensured that any results found are also of value in a practical context. It can be argued that there already have been several case studies in the domain of software product lines, such as those by van der Linden et al. [16], but as stated by Tichy [17], each additional case study is not just a repetition but aims to extend the knowledge previously developed. Our case study proves to be a valuable extension to the body of knowledge already available in that it starts from a '*failed'* project instead of a success story like most of the case studies in this field of research. This failure provided us the perfect opportunity to extract lessons learned valuable to anyone who wishes to undertake a similar project in the future.

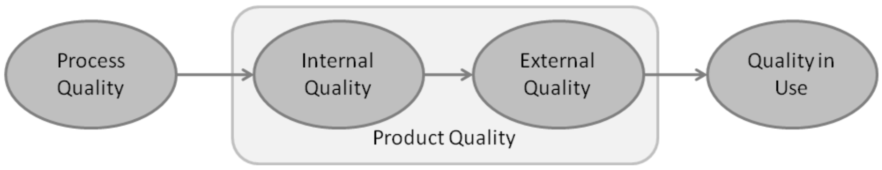

Fig. 1. Categories of Quality in the Product Quality Measurement Reference Model

In order to assess the software product studied in our case, a framework to check the quality of software products was searched for. The ISO/IEC 25010:2011 Software Quality Requirements and Evaluation (SQuaRE) is a series of ISO 9000 compliant standards developed by ISO (www.iso.org), based on the older ISO/IEC 9126 framework and used for evaluating the quality of software products. The Product Quality Measurement Reference Model of the Quality

Measurement Division (2502n) identifies four quality categories of software products related to one another as depicted in Fig. 1. These categories will be used in our case study.

The case study results were processed based on Glaser and Strauss's grounded theory [18]. Starting from a set of open-ended interviews, the transcripts were coded with a qualitative analysis software program to obtain a set of concepts representing the causes of failure of the project according to the interviewees. These concepts were then grouped into categories, on the one hand based on the context of the project itself and on the other hand based on the SQuaRE framework. The project context consisted of a business part, an ICT part and an external (or client) part, and each part was given a separate category in the case specific categorization. The SQuaRE framework categorization corresponds to the quality types defined above. The results of the case study can be found in section five. Before discussing the results, the context of the case is described in the next section.

4. The FinForce case

The case study of this research is based upon a project called FinForce (FF). FF was founded in 2001 as a spin-off company of KBC Group, a European bank and insurance multinational. FF consisted mainly of employees who were transferred from KBC ICT. These employees had, to some extent, experience in the fields of both professional and/or retail payment services. The services FF provided were aimed worldwide towards banks that wanted to outsource their international payment transactions. The business case of FF was based on the fact that international payment transactions were complex, albeit standardized to a certain extent. Before the advent of the Single Euro Payments Area (SEPA) initiative such international transactions were very costly for the banks providing them. Taking advantage of economies of scale, FF would have been able to provide a transaction engine to all banks at a reduced price compared to the in house transaction engines while maintaining a considerable profit margin. The unique selling proposition of FF was being adaptable to any situation possible, hence the tagline in the logo "flexible financial back-office solutions".

The application which was used by FF consisted of four different components, each one with its specific task. There were two main components, called the PE (payment engine) and the PSE (professional service engine), which provided functionalities for respectively the retail and the professional part of the transactions. Besides these main parts, FF was completed with a SIM (smooth interface messaging) component and a TSS (transaction support services) component. The purpose of the SIM component was to provide communication from inside the application towards the external interfaces of the client banks. Thanks to the SIM component client banks could retain their own interfaces. The SIM transformed messages from the FF internal format to the client specific format. The TSS component retained all information needed for the international transactions like e.g. account numbers. Combining these parts provided a complete solution to the client banks for their foreign transactions. The fact that FF consisted of several separate components originated from the fact that although FF was developed from scratch, it was partly based on existing applications inside KBC. Applications like "Pay & Receive" and "Multi-clearing" stood model for respectively the PE and the PSE. An instantiation of the FF application was built for each client with the required adjustments for the interfacing. As such a set of related software systems was created, with only some small differences between the different instantiations.

5. Case study results

To start with, three interviews were conducted with employees who worked for FF, each of them working in a different division, with a different angle on the project. The different angles were carefully selected in order to obtain the most complete overview of FF as possible. To find suitable employees for interviewing, all possible aspects of FF were listed, which resulted in three clearly distinct angles or views, namely the architect or design side view, the business side view and the implementation side view. The setup of the interviews was open ended, as already discussed in the section on research methodology, so the grounded theory approach could be applied afterwards. Some guiding questions were defined, but these merely served the purpose of being some kind of bootstrap. Once the interviews were underway for a couple of minutes, the interviewees continued most of the time out of their own.

During the processing of the three interviews the validity of our choice of angles was enforced as one of the interviewees explained the three organizational pillars of FF, which coincided to a great extent with the views identified a priori. Also some initial observations were made on the causes of failure stated by the interviewees in order to check whether the assumption on the choice of viewing angles could be retained. The main observation was that a significant proportion of the causes of failure were situated in the requirements engineering part of the software engineering process. With this observation the different angles of the interviews were reviewed based on the roles identified in the Volere requirements process [19]. These roles are called supplier (who develops the software), client (who pays for the development), customer (who buys the product after development) and user (who will actually use the software) roles. Besides these, in Volere identified roles a fifth important role also needs to be taken into account. This fifth role is the role of the requirements engineer him/herself. The requirements engineer is responsible for the requirements engineering phase. He has a mediating role between the other roles identified in Volere.

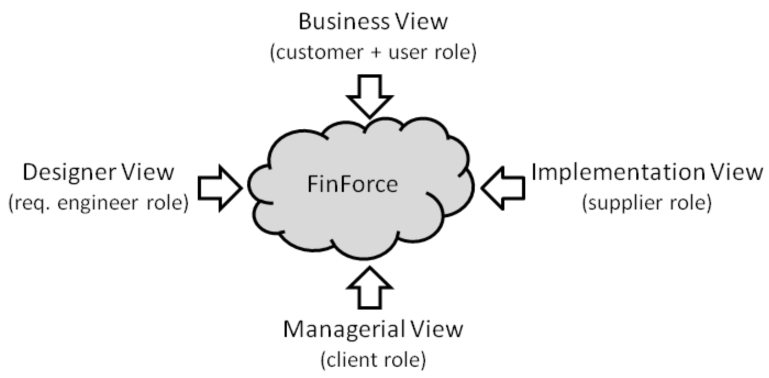

Fig. 2. The different views on FinForce

The five roles as they are identified in the previous paragraph were linked to the viewing angles of the three interviews. The architect's or designer's view in FF can be easily related to the requirements engineer role since it was the designer in FF that was responsible for deciding which requirements to implement. The business side view of FF can be related to the customer role since it was business side that contacted potential customers to relay their requirements to the designers. The business side view can additionally be indirectly related to the user role, because the users are contacted through the customers by whom they are represented. The implementation side view finally can be related to the supplier role as it was the implementation side which was responsible for building the software. Only the client role as identified in Volere cannot be mapped to any of the three views, and subsequently not to any of the three interviews. Therefore it was decided to conduct a fourth interview with someone of FF from a managerial viewpoint as management decided on budget for FF. As such all the roles of Volere are covered, and the assumption that all viewpoints on FF are covered is validated. An overview of the views and the roles can be found in Fig. 2. In the following paragraphs each interviewee's personal opinion will be explored in detail separately before making more general conclusions on the causes of failure for FF.

5.1 The design side view from the technical analyst

The technical analyst consulted did not describe the FF project as a success, but at the same time he stated that "it could not be seen as a complete failure neither". The main problem the technical analyst identified in FF was an issue of scoping. This scoping issue surfaced in several ways throughout the FF case. It started as soon as the requirements engineering was finalized. The requirements were only defined at a very high level of abstraction, without any details. The system needed to be flexible such as to be usable for all possible client banks, but to what extent this flexibility had to be implemented was unclear. FF was created by several analysts simultaneously, but each analyst implemented the variability as he or she saw fit. Because a lack of central coordination or carefully created delimitations in the requirements, one analyst would build in variability with much ardor by making very small components that could be combined in numerous ways while another analyst would build more coarse grained components only leaving a few variability options. At implementation time, the creation of FF took too much time because of the issue of over-flexibility and the related issue of tiny components, as most of the analysts believed that small components were mandatory to make FF as flexible as possible.

A lot of the flexibility that was built into the FF application by the analysts proved to be useless at implementation time. This was also due the fact that the analysts did not possess profound business knowledge. The business side of FF at the same time defined the requirements of FF solely based on former KBC experiences as KBC was the only customer at the start of the project. There was no knowledge on how other banks (potential customers) processed their payment transactions. The technical analyst suggested that it might have been better that some reference visits had been conducted to these other banks in order to have a better view on the variability issues. The problem with such visits is that no bank was very eager to share its way of doing business. A last point made by the technical analyst was that once FF was built and the business had to sell the application, the vendors were too accommodating towards possible customers as they promised them that each of their wishes would be included in FF. Because of this, every time a customer was to be connected, a great deal of adjustments was needed in the FF application.

5.2 The implementation side view from the system administrator

The system administrator interviewed for this case study was mainly responsible for the SIM component of FF and therefore had a good overview on the interfacing issues. According to the system administrator these interfacing problems were plentiful, and moreover they were the single factor that caused each connection of a new customer bank to take much more time than expected. As a result of the interfacing issues, the time period it actually took to connect a new customer bank was 2.5 times bigger than originally estimated. Moreover, according to the system administrator, the problem of non-matching interfaces was a result of bad communication between the representatives of the customer banks and the people who needed to implement the application. After an agreement with a customer bank had been obtained by the vendors, were the same vendors the responsible for the extraction of user specific requirements. Although these user specific requirements were always discussed and extracted by the vendors, the results of these communications were never clearly passed to the people responsible for the actual implementation. Therefore things were wrongly assumed from both sides, what resulted in problems in the mapping of interfaces.

Another somewhat related issue cited by the system administrator is the issue of staff turnover. People who worked on the instantiation for and the connection of a particular customer bank were rarely involved in projects for other banks. Therefore any knowledge developed in the course of one connection was lost most of the time, as this knowledge was mainly tacit of nature. In combination with the fact that there was some kind of unofficial policy to only address problems if they actually occurred even if they could be envisioned up front, made learning from previous experiences hard. The system administrator told us that nowadays a sort of script is developed that can be followed for future instantiations as a way to transfer experience. This script is however only used to connect KBC Group subsidiaries, as FF is only used for subsidiaries nowadays.

5.3 The business side view from the client acquisition officer

The business side of FF was explained to us by one of the client acquisition officers, who were the vendors of FF. Also on the business side there were several issues which impacted the results of the FF project. First of all there was another communication problem. Unlike the communication problem between implementers and customers described earlier, this problem was situated between the customer banks and the customer acquisition team from FF. Potential customer banks were approached by the customer acquisition team and they proposed to the banks the FF business model as it was seen and implemented by ex-KBC staff. The difficulty herein was that potential customers had to understand the KBC oriented context in which the business model was written, and link this to their own business model. If customers had to adapt to another context in order to be able to state their own needs, communication got difficult at least or customers even could altogether lose their interest in FF.

Another serious and similar obstacle in obtaining customers was the KBC background of FF. Potential customer banks were not fond of disclosing sensitive information to what they believed a competitor in the bank market. This is best illustrated with a small anecdote from the interviewed client acquisition officer. He told us: “When we approached banks and we wanted to give a presentation, upon booting our laptops the first thing appearing on the screen would be a giant KBC logo, as our equipment came from KBC ICT. Clients would ask straight away what the logo meant, and the first seeds of distrust would be sown”.

5.4 The managerial side view from the Chief Executive Officer

The interview with the former Chief Executive Officer (CEO) of FF gave a good overview on how FF came to be, since the interviewee was involved in FF from cradle to grave. He described that while in the end FF was incorporated back into KBC, some benefits still remain present in KBC. Since it was the objective to sell FF towards external clients, it was necessary to document everything as clearly as possible. This extensive documentation still proves to be valuable to KBC. Besides this documentation advantage there were no further major benefits according to the CEO. The issues concerning FF on the other hand were plentiful. One of the main reasons of failure according to the CEO was that standardization was not part of the corporate culture of any bank. Apart from regulations that are imposed by higher authorities, there was no way that all banks would agree that FF was providing the standard way of working. The fact that the focus of FF shifted constantly during the project only added to the confusion and the diversity of the application. For example, at one point in time SEPA-transactions were to be included, while at another point in time they had to be excluded at all costs.

The CEO furthermore mentioned that one of the most characteristic properties of FF was that it provided a very rich and diverse set of functionalities, which seems to be in direct contrast with the will to standardize. The existence of such a contrast was only possible due to an unfocused way of working. Each time some functionality was created it was just added to the complete system without looking at a way to make the new functionality fit to the rest of the system. This led to a system with enormous possibilities which could never be fully used due to the fact that the system was one big clutter.

5.5 Global results of the case study

The previous paragraphs shed some light on interesting particularities for each of the views on the project, but nevertheless the overarching concern relates to the way variability was addressed during requirements engineering. As already mentioned, a big problem was that there was variability on certain places in FF where it should not have been, and there were places where there was no variability but where it was needed. This resulted from a flawed vision on the system, and as much from poor requirements engineering. According to the higher management who decided to create the FF spin-off company, FF was meant to be the application that would set the standard for all international payments transactions. Pre-studies conducted in corporation with renowned consulting firms spoke of worst case scenarios requiring at least 30 client banks within three years. It can be argued that the crisis around 9/11 had an unpredictable impact on the business case, but nevertheless the order of magnitude envisioned was in sharp contrast with the final number of external banks that were connected to FF before the project got drastically clipped in 2007-2008. The overly positive attitude towards FF created the need that FF had to be able to work for everyone and as soon as possible. The broad scope required loads of variability, while the short time span made it impossible to think the project through thoroughly before starting to develop the software system. Apart from some most basic requirements gathering, the requirements engineering phase was thus more or less completely skipped. Variability was implemented without any guidelines. The result was that the application was capable to handle a very broad scope, but at the same time making one instantiation for a particular client took so much time to develop that the whole system’s costs were way higher than it could possible gain as benefits by its broad scope.

In order to perform a more systematic analysis of the root causes of failure, all the 51 causes of failure mentioned in the interviews were listed and categorized as explained in section 3. The results from this categorization are to be found in table 1. Some of the failure cause extracted from the interviews could not easily be categorized as can be seen in the "Various" row in table 1. These 15 concepts actually overlap the “Business” and “ICT” categories in the categorization. Looking at the SQuaRE based categorization; we see that 7 concepts do not really categorize themselves into a certain quality. These concepts state facts that do not really have a quality aspect to them, therefore they are labeled “No Quality”.

Table 1. The failure concept categorization

	Process Quality	**Internal Quality**	**External Quality**	**Quality in Use**	**No Quality**	**Total**
Business	9	0	3	0	3	**15**
ICT	10	1	2	0	0	**13**
External	2	0	1	3	2	**8**
Various	5	1	4	3	2	**15**
Total	**26**	**2**	**10**	**6**	**7**	**51**

A lot of the failure concepts are process quality related. Inside the ICT category even around 75% of the failure concepts are categorized as process quality. This leads to the conclusion that the quality of the development process should receive enough attention in order to be more successful. Business and ICT have about the same importance as they account for almost the same amount of failure concepts. This validates that business and ICT should work together as equal partners in order to have success in software projects like the FF project. The need for a focus on process quality along with the need for a good way to deal with variability during requirements engineering led to the development of a theoretical framework describing the issues at hand from the observations made in the FF case. The resulting framework is however usable in any context were (sets of) software systems are made and variability is significantly present. The framework described in the next section provides a focus that is not only useful in the development of such systems, but that should be reasoned about explicitly in order to evade failures like the ones present in FF.

6. The Harmonization and Variabilization framework

The proposed framework consists of two central decisions key to variability management in requirements engineering and is explained with the Volere roles in mind. The first decision concerns the harmonization of requirements. The user, client and customer stakeholders within a certain software system context form the demand side of that software system. Once all the requirements of the demand side have been elicited, an analysis of the (dis)similarity of these requirements can start. Some requirements can be the same for everyone (the common part, or the commonality), while other requirements can differ between demand side members (the variable part, or the variability). It is up to the requirements engineer to find the similarities and differences in the requirements and to confront the clients, customers and users with these (dis)similarities.

As such, harmonization is much more than just requirements gathering and elicitation. At this point a requirements engineer can attempt to convince users, through the customers, to assimilate each other's requirements so that the amount of variability that needs to be dealt with is reduced. One can start from the similarities in the requirements and try to expand these by convincing users that they do not need the differentiating parts of the requirements. The supplier and customer can, for example, attempt to reduce variability by trying to set standards. This balancing is not straightforward as each user wants the to-be-created software system to be tailored as much as possible to his personal context, while at the same time the client, who is the stakeholder that pays for the development, support and maintenance of the to-be-created software system will rather be in favor of diminishing the variability in order to cut down the costs of development, support and maintenance. The customer will also be affected by the amount of variability. He will be charged a price by the client based on the costs made by the client. Too much variability holds the risk of a too high price for the customer. The amount of variability within the set of all users' requirements should thus not only be identified, but also be managed as well during the harmonization.

The second decision concerns the variabilization of requirements, and can be done once the harmonization is finished. The amount of variability originating from the diverging demand side's viewpoints cannot be altered anymore, but the support of this variability by the to-be-created software system is still to be decided upon. In the case of a supplier developing software for multiple customers, the client may decide that some variability will be taken into account into a project's requirements problem, while some variability (and the corresponding requirements) will be left aside for the customer to deal with in the specific contexts of the users.

Fig. 3 visualizes the harmonization (1) and variabilization (2) as setting boundaries within an octagon representing the union of all users' requirements. The harmonization divides the requirements into a part which is common for all users (commonality) and a part which differs for the users (variability). The variabilization further divides the variability part into a part that is sufficiently shared and will therefore be supported by the client (shared variability), and a part which is too specific to justify investments by the client and will thus be left to the customer to deal with (specific variability). Both harmonization and variabilization should be seen as a careful balancing exercise with the amount of requirements' variability at stake. The octagonal shape is used for the total set of demand side stakeholders' requirements to represent a sense of unlikelihood that the boundaries drawn by harmonization and variabilization are completely located at the edge of the requirements set. This would mean that no division in requirements is made by the harmonization and/or the variabilization, which is in most cases not optimal.

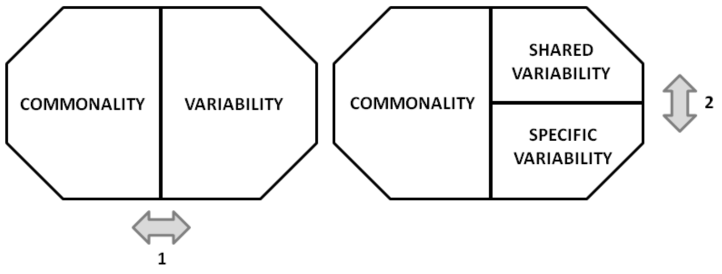

Fig. 3. The harmonization (1) and variabilization (2)

In the FinForce case, no explicit decisions were taken about harmonizing requirements or about variabilization. This led to a situation where there was little commonality and all variability was considered to be shared, or stated in terms of drawn boundaries, they were both drawn at the "edge of the octagon". As a result, the development cost of the software was raising high on the client's side, and moreover the cost for implementing the software at the users' side was raising high as well because of the huge amount of shared variability that needed to be dealt with during each implementation. Should both variability decisions as defined here have been taken consciously during the development of FinForce, the amount of possible causes of failure probably would have been reduced significantly.

7. Conclusion

Variability management is a complex process. The FinForce interviewees all mention this as causes of failure. As such, variability should be considered during the whole development project of a software system, especially when the system's success is critically impacted by variability management, such as in software product lines. The case study showed that it is not only a matter of creating a software product that supports variability, but that every bit of variability should be considered carefully from the requirements engineering phase on. When quality of the requirements engineering process is inferior, the product will not deliver the expected benefits. The decisions concerning the variability should be taken with due attention. Only when all parts of a company, both business and ICT, are ready to deal with variability, this variability management has the chance to succeed. Communication is of critical importance between users, implementers and developers as variability manifests itself in small details. Leaving room for interpretation can be fatal, as literally mentioned by one of the interviewees. Although it is the product that will be used eventually, it is the development process of the product where most problems are situated. This chimes with Deming [20] who states that the most effective way to improve the product quality is to improve the quality of the process that produces the product. For software systems that need to be flexible through variability, this may well be the only way to reach quality.

Based on the observations and knowledge extracted from the case study a framework was developed that describes the variability decisions that need to be made during the requirements engineering phase. The harmonization and the variabilization are the two decisions that should be taken with due attention in order to prevent the software system to come into troubles with the variability. First it must be decided how much variability should be present in the software system. After this, it must be decided whether the needed variability will be supported by the software system that is being created. When these two decisions are taken, a lot of issues, like the ones observed in FinForce, can possibly be avoided as the scope of the project would have been clear. As such the framework can provide the needed knowledge transfer channel towards practice in order to avoid tripping over the pitfall of variability.

References

[1] T. Huysegoms, M. Snoeck and G. Dedene, "Building a Requirements Engineering Methodology for Software Product Lines," in *5th SIKS/BENAIS Conference on Enterprise Information Systems*, Eindhoven, Nederland, 2010, pp. *25-34*.

[2] K. C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak and A.S. Peterson, "Feature-Oriented Domain Analysis (FODA): feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

[3] K. Schmid and I. John, "A customizable approach to full lifecycle variability management," *Science of Computer Programming*, vol. *53*, no. *3*, pp. *259-284*, December, 2004.

[4] S. D. Kim, S. J. Her and S. H. Chang, "A theoretical foundation of variability in component-based development," *Information and Software Technology*, vol. *47*, no. *10*, pp. *663-673*, July, 2005.

[5] S. Liaskos, S. A. McIlraith, S. Sohrabi and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering," *Requirements Engineering*, vol. *16*, no. *3*, pp. *227-249*, September, 2011.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. *5*, no. *1*, pp. *143-168*, January, 1998.

[7] C. Atkinson, J. Bayer and D. Muthig, "Component-Based Product Line Development: The KobrA Approach," in *First Product Line Conference*, Denver, Colorado, 2000, pp. *289-309*.

[8] M. Sinnema, S. Deelstra, J. Nijhuis and J. Bosch, "COVAMOF: A framework for modeling variability in software product families," in *Third Software Product Line Conference*, Boston, Massachusetts, 2004, pp. *197-213*.

[9] K. Pohl, G. Böckle and F.J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, 1st ed., New York, New York: Springer-Verlag, 2005.

[10] L. Chen, N. Ali Babar and N. Ali, "Variability management in software product lines: a systematic review," in *13th International Software Product Line Conference*, San Francisco, California, 2009, pp. *81-90*.

[11] P. Clements and L. Northrop, *Software product lines: practices and patterns,* 3rd ed., Boston, Massachusetts: Addison-Wesley, 2001.

[12] Software Engineering Institute, Carnegie Mellon. (2013, February). [online]. Available: http://www.sei.cmu.edu/productlines/

[13] D. L. Moody, "Using the world wide web to connect research and professional practice: Towards evidence-based practice," *Informing Science Journal*, vol. *6*, no. *1*, pp. *31-48*, January, 2003.

[14] H. Kaindl, S. Brinkkemper, J. A. Bubenko, B. Farbey, S. J. Greenspan, C. L. Heitmeyer, J. Sampaio do Prado Leite, N. R. Mead, J. Mylopoulos and J. Siddiqi, "Requirements engineering and technology transfer: Obstacles, incentives and improvement agenda," *Requirements Engineering*, vol. *7,* no. *3*, pp. *113-123,* September, 2002.

[15] R. K. Yin, *Case study research: design and methods*, 4th ed., Thousand Oaks, California: Sage, 2002.

[16] F. Van der Linden, K. Schmid and E. Rommes, *Software product lines in action: the best industrial practice in product line engineering*, 1st ed. Berlin, Germany: Springer, 2007.

[17] W. F. Tichy, "Hints for Reviewing Empirical Work in Software Engineering," *Empirical Software Engineering*, vol. *4*, no. *4*, pp. *309-312,* December, 2000.

[18] B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*, 1st ed. Chicago, Illinois: Aldine Publishing, 1967.

[19] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 2nd ed., Boston, Massachusetts: Addison-Wesley, 2006.

[20] W. E. Deming, *Out of the crisis*, 1st ed. Cambridge, England: MIT Press, 2000.

Biographical notes

Tom Huysegoms
Tom Huysegoms is a Ph.D. student in computer science at the Katholieke Universiteit Leuven in the Department of Decision Sciences and Information Management of the Faculty of Business and Economics. His research focuses on requirements engineering and variability management and is done in collaboration with KBC, an international bank and insurance company active in East and Central Europe.

www.shortbio.net/tom.huysegoms@kuleuven.be

Monique Snoeck
Monique Snoeck holds a Ph.D. in computer science from the Katholieke Universiteit Leuven. She is full professor in the Department of Decision Sciences and Information Management of the Faculty of Business and Economics of the Katholieke Universiteit Leuven and visiting professor at the Facultés Universitaires Notre Dame de la Paix, Namur. Her research focuses on conceptual modeling, requirements engineering, software architecture, model-driven engineering and business process management.

www.shortbio.net/monique.snoeck@kuleuven.be

Guido Dedene
Guido Dedene is a professor in management information systems in the Department of Applied Economics at the Katholieke Universiteit Leuven, Belgium. His lecturing and research activities concentrate on formal methods for systems development and quantitative systems management techniques. For five years, Prof. Dedene was also director of European services for GUIDE and SHARE Europe, an association of over 2,000 mainframe users.

www.shortbio.net/guido.dedene@kuleuven.be

Antoon Goderis
Antoon Goderis is an enterprise architect at KBC Global Services. He has a PhD in Computer Science from The University of Manchester.

www.shortbio.net/antoon.goderis@kuleuven.be

Frank Stumpe
Frank Stumpe holds a Ph.D. in project management in complex-cybernetic systems. He worked as consultant in project and change management and was lecturer at the RWTH Aachen for Project Management and holds different guest lecturers. The professional career of Frank Stumpe was international and cross industrial. Starting from managing European research projects he works for heavy industry, food industry as well as hotels and transport. He was CEO of IPS Bulgaria Ltd. and member of the Group-ExCo responsible for training and methods. Today working as head of the competence centre project based working at KBC Group.

www.shortbio.net/frank.stumpe@kuleuven.be

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X

Available online at www.sciencesphere.org/ijispm

HCist '2013 Lisbon, Portugal

Conference on Health and Social Care Information Systems and Technologies

HCist'2013 - International Conference on Health and Social Care Information Systems and Technologies, intends to gather Healthcare Information Systems and Technologies professionals and academics to share and discuss current challenges, developments, case studies, integrated and practical solutions, as well as new products, findings and approaches to leverage the use of Information Systems and Technologies in healthcare.

Researchers are invited to submit their manuscript electronically at the conference webpage until May 15, 2013.

General Chairs:	Ricardo Martinho, Polytechnic Institute Leiria, Portugal
	Rui Rijo, Polytechnic Institute Leiria, Portugal
Program Chair:	Joseph Tan, McMaster University, Canada

Detailed information available at: hcist.eiswatch.org

ADVERTISING

SciKA

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm