Chapter VI
Requirements Engineering: A Review of Processes and Techniques

Fernando Flores
Autonomous University of Aguascalientes, Mexico

Manuel Mora
Autonomous University of Aguascalientes, Mexico

Francisco Alvarez
Autonomous University of Aguascalientes, Mexico

Rory O'Connor
Dublin City University, Ireland

Jorge Macias-Luévano
Autonomous University of Aguascalientes, Mexico

ABSTRACT

Requirements engineering is the process of discovering the purpose and implicit needs of a software system that will be developed and making explicit, complete, and non ambiguous their specification. Its relevance is based in that omission or mistakes generated during this phase and corrected in later phases of a system development lifecycle, will cause cost overruns and delays to the project, as well as incomplete software. This chapter, by using a conceptual research approach, reviews the literature for developing a review of types of requirements, and the processes, activities, and techniques used. Analysis and synthesis of such findings permit to posit a generic requirements engineering process. Implications, trends, and challenges are then reported. While its execution is being mandatory in most SDLCs, it is done partially. Furthermore, the emergence of advanced services-oriented technologies suggests further research for identifying what of the present knowledge is useful and what is needed. This research is an initial effort to synthesize accumulated knowledge.

INTRODUCTION

In the field of software engineering several process models have been formulated to guide the development of software systems (e.g., software or system development life-cycle). Independent of what process model is selected by a development team, all activities conducted can be grouped into three main macro-phases: **s*ystem definition*** (*software specification of functional and constrain requirements),* ***system development*** *(design and building),* and ***system deployment*** *(software implementation, software validation (to confirm that the new software system satisfies the users' needs)* and *software evolution (evolution of the users' requirements as the users' reality evolves))* (Sage & Armstrong, 2000; Sommerville, 2002).

First, macro-phase's activities have been studied by the requirements engineering (RE) discipline, which can be defined as: "the process of discovering the purpose of the software system by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation" (Nuseibeh & Easternbrook, 2000). The overall goal of RE is to elicit valid users´ requirements because the strong impact on quality and cost of the final software product. Accordingly to Jin et al (1998): "…errors made at this stage are extremely expensive to correct when are discovered during testing or during actual working." However, even though such evidence of relevance and that RE has been identified (Sommerville, 2005) as essential for successful software development, these activities are often overlapped, uncompleted, or missed in development projects. As Sumano (1999) alerts "it is a general practice not to do it well, or do it faster and careless, because they do not have enough time or because they do not know a good methodology to do it." Consequently, it is possible multiple errors are introduced in early activities and not discovered until later phases of the lifecycle raising the project costs and exceeding the project deadlines.

In this chapter, we use a conceptual research methodology (Glass, Vessey, & Ramesh, 2002; Mora, 2004) to review the state of the art on the process and techniques used in software requirements engineering for software products to answer the following research questions: (a) How can the software requirements be classified?, (b) How can the main processes, activities, and techniques proposed by the software requirements engineering, be organized ?, and (c) Can these processes, activities, and techniques be synthesized in a theoretically-developed generic process of software requirements engineering? According to Mora (2004), despite several sources report the utilization of a conceptual research approach and its wide usage in the domain of the software engineering (43%) (Glass et al., 2002), there is little detailed literature on how to use this research method. Counelis (2000) quoted by Mora (2004) indicates that conceptual research is part of the research methods that study ideas, concepts or constructs on real objects rather than study them directly. Despite scarce literature, Mora (2004) reports that several studies consider the conceptual research method as common as the survey, experimental, and case study methods. This chapter then uses the process described in Mora (2004) that consists in the following phases: (1st) formulation of the research problem; (2nd) analysis of related studies; (3rd) development of the conceptual artifact; and (4th) validation of the conceptual artifact. The first phase and second phases are similar to other research methods. In the third phase, two activities are conducted: the development of a high-level framework/model and the development of low-level details of specific components selected from the high-level framework/model. This third phase is a creativity-intensive process guided by the findings, contributions, and limitations found in the second phase and a set of preliminary proforms that are fixed through an iterative process (Andoh-Baidoo, White, & Kasper, 2004). In the last phase, the conceptual artifact's validation is developed through: face validity from a panel of experts, logical argumentation, or proof of concept developing a prototype or pilot survey.

The objectives of this research are: (a) to develop an updated classification of software requirements,

(b) to clarify the similarities and differences among software requirements engineering methods reported in the literature, and (c) to identify a generic process for software requirements engineering trough a synthesis process. This research is strongly motivated by the lack of discussion in the requirements engineering literature about standardized (e.g., generic) software requirements engineering stages and activities. Accordingly, academics and practitioners face a myriad of process and techniques and organizational utilization is influenced rather for knowledge availability than by system's adequacy. Because every software development lifecycle starts with a software definition, this study contributes to improve our understanding and application of general software development lifecycles through identifying a generic method for software requirements engineering.

BACKGROUND

The concepts of *requirement* and *requirements engineering* are core for the software engineering discipline. From the multiple definitions of what a requirement is, two comprehensive definitions (IEEE software engineering glossary, Abbott, 1986) are reported in a relevant source (SEI Curriculum Module SEI-CM-19-1.2, 1990): (a): *(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component*; and (b) *requirements is any function, constraint, or other property that must be provided, met, or satisfied to fill the needs of the system intended user(s).*

In turn, the *requirements engineering* can be defined as a discipline or knowledge area (Zave, 1997; SWEBOK, 2004; Sawyer & Kotonya, 2000; Gonzalez, 2005) and as an abstract or specific process (SEI Curriculum Module SEI-CM-19-1.2, 1990; SWEBOK, 2004; Nuseibeh & Easternbrook, 2000; Sawyer & Kotonya, 2001; Sommerville, 2005). As discipline, Zave (1997) defines requirements engineering as: "the branch of software engineering concerned with the real-world goals for functions and constraints on software systems." In same study, the author states that "the great difficulty in constructing such a classification scheme is the heterogeneity of the topics usually considered part of requirements engineering." These topics include the following: tasks that must be finished, problems that must be solved, solutions to problems, ways of contributing to knowledge, and types of system. In turn, in SWEBOK (2004), the requirements engineering knowledge area is "...concerned with establishing a common understanding of the requirements (e.g., study of methods for) to be addressed by the software product." In Sawyer and Kotonya (2001), requirements engineering is considered as "the knowledge area (that) is concerned with the acquisition, analysis, specification, validation and management of software requirements." For Gonzalez (2005), requirements engineering is the study of "methods for capturing, specifying, and managing requirements." Under such definitions and research effort classification, this study can be considered as a conceptual research on the tasks must be performed (capturing, specifying, communication/validation, and managing) and can be located also in the convergence of the RE and software requirements engineering research streams (SWEBOK, 2004).

As an abstract process the requirements engineering "... consists of a set of transformations that attempt to understand the exact needs of a software-intensive system and convert the statement of needs into a complete and unambiguous description of the requirements, documented according to a specified standard" and define the activities of "requirements elicitation, analysis, and specification" (SWEBOK, 2004). In similar mode, RE as a broad process can be defined as "the process of discovering the purpose of the software system by identifying stakeholders

and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation" and can be composed by the following activities: *eliciting requirements, modeling and analyzing requirements, communicating requirements, agreeing requirements,* and *evolving requirements* (Nuseibeth & Easterbrook, 2000).

As a specific process, the SEI curriculum module SEI-CM-19-1.2 (1990) proposes the following activities: requirements identification, identification of software development constraints, requirements analysis, requirements representation, requirements communication, and preparation for validation of software requirements. For Sawyer and Kotonya (2001), requirements engineering must have the following activities: requirements elicitation, requirements analysis, requirements specification, requirements validation, and requirements management. Sommerville (2005) defines requirements engineering as an abstract process as "... a structured set of activities that help develop this understanding and that document the system specification for the stakeholders and engineers involved in the system development," and as a specific process, as composed of the: elicitation, analysis, validation, negotiation, documentation, and management activities. Other proposal for requirements engineering activities (from the ESA Software Engineering Standards Issue 2, prepared by ESA Board for Software Standardization and Control, 1994), proposes a differentiation from user and software: user requirement (capture the user requirements, determination of operational environment, specification of user requirements, and reviews) and software requirement (construction of the logical model, specification of software requirements, and reviews).

Hence, despite the literature reporting multiple processes for requirements engineering, there is not a unique and agreed (or standardized) requirements engineering process, but some shared activities can be identified. Then, given the vast literature and myriad of definitions, process, and techniques, their understanding and final utilization by academics and practitioners is obfuscated.

MAIN FOCUS OF THE CHAPTER

The main focus of this chapter is to provide an updated and comprehensive software requirements classification, an organized view of processes, activities, and techniques for software requirements engineering and identify core activities and techniques for positing a generic process for requirements engineering. The contribution is to improve the understanding of the requirements engineering process, activities, and techniques. The conceptual analysis is realized through the development and utilization of a set of pro-forms (Andoh-Baidoo et al., 2004). Units of study are the process, activities, and techniques discussed in the main papers reported in the literature.

A requirement can be defined as a mandatory or wished attribute (as an adjective), capability (as a verb), or condition (as a logical or numerical constrain) that a product, service, process, or system must possess. While requirements are characteristics owned by artifacts or systems (in the software domain), these are demanded by human beings (e.g., all stakeholders related with the definition of the system). Then, a requirements engineering process for determining the set of valid requirements for a system can be considered a human-intensive interaction process. Furthermore, while an extensive research (Beckworth & Garner, 1994; El-Eman & Madhavjin, 1995; Nikula, Fajaniemi, & Kalviainen, 2000; Juristo, Moreno, & Silva, 2002; Neil & Laplante, 2003) has been conducted on process, techniques, and their real utilization in organizations, few studies have been focused in classifying requirements and the findings show overlaps, omissions, and mixed interpretations. Then, for achieving the research purpose implicitly established in research question (a), we believe that a comprehensive and updated requirements classification is needed. Table 1 shows such classification

from several sources (Brackett, 1990; ESA PSS-05-03, 1995; SWEBOK, 2004) analyzed.

Main findings from Table 1 are: (a) the identification of the environmental requirements, few mentioned and explained in usual literature; (b) the sub-classification and focus of external requirements on social and human affairs; and (c) the re-grouping of classic functional versus non-functional requirements with emergent relevant sub-types such as: security, lifecycle, inverse, and documentation requirements. In particular, the security issues are not reviewed extensively in this study but we recognize as the information systems are used for mission critical systems (and supported by the software systems), and deployed in ICT internet-based platforms, this issue can be critical. This new classification suggests that software requirements systems engineers should not omit the social and human influences that the external politic-power, socio-cultural, legal, and economic environmental systems perform on the organization and lately of the software systems users. While that information systems literature (Keen) has extensively alerted on such issues, the

Table 1. An updated and comprehensive software requirements classification

Main Category	Type	Sub-type
ORGANIZATIONAL ENVIRONMENT These requirements can be derived from the internal and external environment where the software system will be deployed.	**EXTERNAL ENVIRONMENT** Requirements derived from the potential influence of the relevant social systems in the outer organizational environment for the software system	• **Politic-power requirements** • **Legal requirements** • **Socio-cultural requirements** • **Economic requirements**
	INTERNAL ENVIRONMENT Requirements derived from the potential influence of the relevant social systems in the inner organizational environment for the software system	• **Business goals requirements** • **Organizational interface requirements** • **User's expectation requirements**
FUNCTIONAL These requirements can be derived for defining the behavioral capabilities for the software system to be deployed.	**SERVICES**	The visible capabilities that the software system will provide to users (humans and other systems)
	TECHNICAL INTERFACES	The characteristics that permit to the software system communicating with other systems
	OPERATIONAL	The characteristics that define how to operate correctly the software system
NONFUNCTIONAL These requirements can be derived for defining a complementary set of characteristics (usually required for user's system acceptance) to the behavioral capabilities for the software system to be deployed.	**PERFORMANCE**	The characteristics that define how well to the software system operates
	CONSTRAINS Requirements derived to establish explicitly conditions (events that must or must not occur) to be satisfied by the software system	• **Inverse** — Requirements on what must not occur in the software system • **Lifecycle** — **Design, Building, Testing, User Implementation, Disposal Requirements**
	QUALITY Classic requirements for the software system that jointly define its objective and subjective quality (e.g. overall conformance to user's expectations)	**Functionality, Reliability ,Usability Efficiency, Maintainability, Portability Requirements** (based in ISO 9126)
	SECURITY	Requirements derived to establish explicitly the security (e.g. safeness, protection, vulnerability, recoverability, survivability) characteristics for the software system
	DOCUMENTATION These requirements found out the documentation needs.	**Technical, Operative, Politics & Procedures**

Table 2. List of software requirements engineering techniques/methods analyzed

Technique/Method	Sources
Interview	Durán and Bernárdez (2000)
JAD (Join Application Development)	Satzinger et al (2000)
SADT (Structured Analysis and Design Technique)	Siltala (2004)
Structured System Analysis	ITC InfoTech (2006)
Object Oriented Modeling	Insfrans et al (2001)
Goal Oriented Modeling	Delor et al (2003); Mylopoulos et al, 1999
Prototype	Buchenau and Fulton (2000)
Walkthroughs[1]	Bias (1991)
PSL / PSA	Teichroew and Sayani (1980); Beregi (1984);
Formal Methods	FAA System Safety Handbook (2002); Vienneau (1993)
Artificial Intelligence based Techniques	Pohl et al (1994)

software engineering literature is scarce. However, some authors (Ross, 1997; referenced by Brackett, 1990; Mylopoulos, Bordiga, Jarke, & Koubarakis, 1999) have identified the relevance of considering such issues (e.g., requirements should not only answer the what and general how questions, but also the why inquiry).

Table 1 does not imply that each software system to be developed should consider all types and sub-types. Rather than, this suggests that requirements engineers should analyze what of them are relevant for such specific situations and decide jointly with stakeholders what will be considered.

For achieving the second purpose on the clarification of similarities and differences among the several software requirements engineering processes, activities, and techniques reported in the literature, several pro-forms are used. Table 2 shows the techniques/methods analyzed. In Table 6 the four processes analyzed are showed.

The pro-form in Table 3 is used to identify the inputs, phases, and results proposed by a software engineering process analyzed. In this table, the software requirements engineering process posed by SEI is conceptually dissected. A similar analysis was conducted with the remainder process reported in Table 6.

Simultaneously, for a better understanding, a detailed analysis of each technique/method was conducted. Table 4 shows the pro-form used to identify their name, description, tasks, discipline that belongs to, a classification, and the sources. In Table 4, SE stands by system engineering, IS by information systems, and SwE by software engineering.

From the analysis conducted to techniques and process, two main general findings can be reported: (a) a re-grouping of techniques/methods (Table 5) and (b) a comparative of software requirements engineering process (Table 6). Techniques can be classified into four classes: traditional, group oriented, modeling oriented, and formal logic (Table 5).

The first class (traditional) can be used to the contextual analysis and elicitation activities for its potential for managing social-politic and human affairs. The second class (group oriented) can be used in the elicitation, constraints identification, and metric parameter definition activities for its clarity of representation for physical artifacts. The third class (modeling oriented) can be used to define data and processes representations, and finally, the fourth class (formal logic) to do elicitation, modeling, and validation activities when mission critical and

Table 3. Analysis of a software requirements engineering

Software Requirements Engineering Process from SEI			
Purpose : It is concerned with the definition of software requirements –the software engineering process of determining what is to be produced- and the products generated in that definition			
Inputs	**Proposal Phases**		**Results**
	Name	**Description**	
Context analysis should answer the following questions: • Why is the software developed? • What is the environment where the software will be created and operated? • What are the technical, operational, and economic boundary conditions that an acceptable software implementation must satisfy?	**Requirements Identification**	The software requirements are elicited from people or derived from systems requirements.	There are three principal groups: • **Functional Requirements**: these specify the functions of the system or system component. • **Non-Functional Requirements**: these consider the performance, interface, and reliability requirements. • **Constraints on the design**: these define the operational and implementation limits.
	Identification of Software Development Constraints	Acceptable constraints (costs, the characteristics of the hardware to which the software must interface, existing software with which the new software must operate, fault tolerance objectives, and portability requirements and implemented within the restrictions imposed) are identified.	
	Requirements Analysis	Potential problems, classification of requirements and evaluation of feasibility and risk are developed.	
	Requirements Representation	The results of the requirements identification are portrayed.	
	Requirements Communication	The results of requirements definition are presented to diverse audiences for review and approval.	
	Preparation for Validation of Software Requirements	The criteria and techniques are established for ensuring that the software, when produced, meets the requirements. The customer and software developers must reach agreement on the proposed acceptance criteria and the techniques to be used during the software validation process, such as execution of a test plan to determine that the criteria have been met.	

Table 4. Technique/method analysis

Technique/Method: Formal					
Description: Formal Methods (FM) consists of a set of techniques and tools based on mathematical modeling and formal logic that are used to specify and verify requirements and designs for computer systems and software. Formal methods may be used to specified and model behavior of a system and to mathematically verify that the system design and implementation satisfy system functional and safety properties. These specifications, models, and verifications may be done using a variety of techniques and with various degrees of rigor.					
Tasks: **Level 1:** Formal specification of all or part of the system. **Level 2:** Formal specification at two or more levels of abstraction and paper and pencil proofs that the detailed specification implies the more abstract specification. **Level 3:** Formal proofs checked by a theorem prover.					
Discipline			**Classification**	**Reference:**	
SE	**IS**	**SwE (X)**	Formal Logic	[FAA2000]	FAA System Safety Handbook, Appendix D; (2000) "Appendix D Structured Analysis and Formal Methods"

Table 5. Classification of software requirements engineering techniques/methods

Class	Technique
Traditional	Interview
Group Oriented	JAD (Join Application Development)
	SADT (Structured Analysis and Design Technique)
	Walkthroughs
Modeling Oriented	PSL / PSA (Problem Statement Language / Problem Statement Analyzer)
	Structured System Analysis
	Object Oriented Modeling
	Goal Oriented Modeling
	Prototypes
Formal Logic	Formal Techniques
	Artificial Intelligence based Techniques

Table 6. Comparison of software requirements engineering processes

Documents / Generic Process Stages	Software Requirements SEI Curricula: SEI (1990)	Requirements Engineering: A Road Map: Neseibeth & Easterbrook (1990)	Software Requirements (in SWEBOK): Sawywer & Kontoya (2001)	Guide to the Software Requirements Definition Phase: ESA Board for Software Standardization and Control; European Space Agency (1990)
A.1 Social Context Analysis	Briefly mentioned	---	---	---
A.2 Operational Context Analysis	Briefly mentioned	Briefly mentioned	Briefly mentioned	Operational environment determination
A.3-4 Elicitation and Analysis	Requirements Identification	Eliciting Requirements	Requirements Elicitation	Definition of the user requirements
	Identification of software development constraints	Modeling and analysis requirements	Requirements Analysis	
	Requirements Analysis			
A.5 Requirements modeling and representation	Requirements Representation		Requirements Specification	Specification of the user requirements
A.6 Requirements Communication	Requirements Communication	Communicating Requirements		---
A.7 Requirements Validation & Specification	Preparation for validation of software requirements	Agreeing Requirements	Requirements Validation	Reviews
A.8 Changing Management	---	Evolving Requirements	Requirements Management	---

high-risk process are being modeled. Table 6 shows a comparison of the several software requirements engineering processes analyzed.

It is important to note that the term "elicitation" includes the normal "gathering information" and the "analysis of the information gathered" that requirement engineers have to perform in order to find all the issues that could be potentially useful for determining the characteristics mandatory and expected of the software system. These issues can come from real organizational or user's events or situations but the users could omit intentional or involuntary because political-power reasons or simply by daily routine. Findings from Table 7 suggest that social context as well as operational context analysis are critical activities to be pursued by software requirements engineering in order to avoid critical organizational or user's omissions. Based in such a comparison and using a well-known notation for process specification in systems engineering (IDEF0, Mayer, 1990), Figure 1 shows the posited "generic software requirements engineering process."

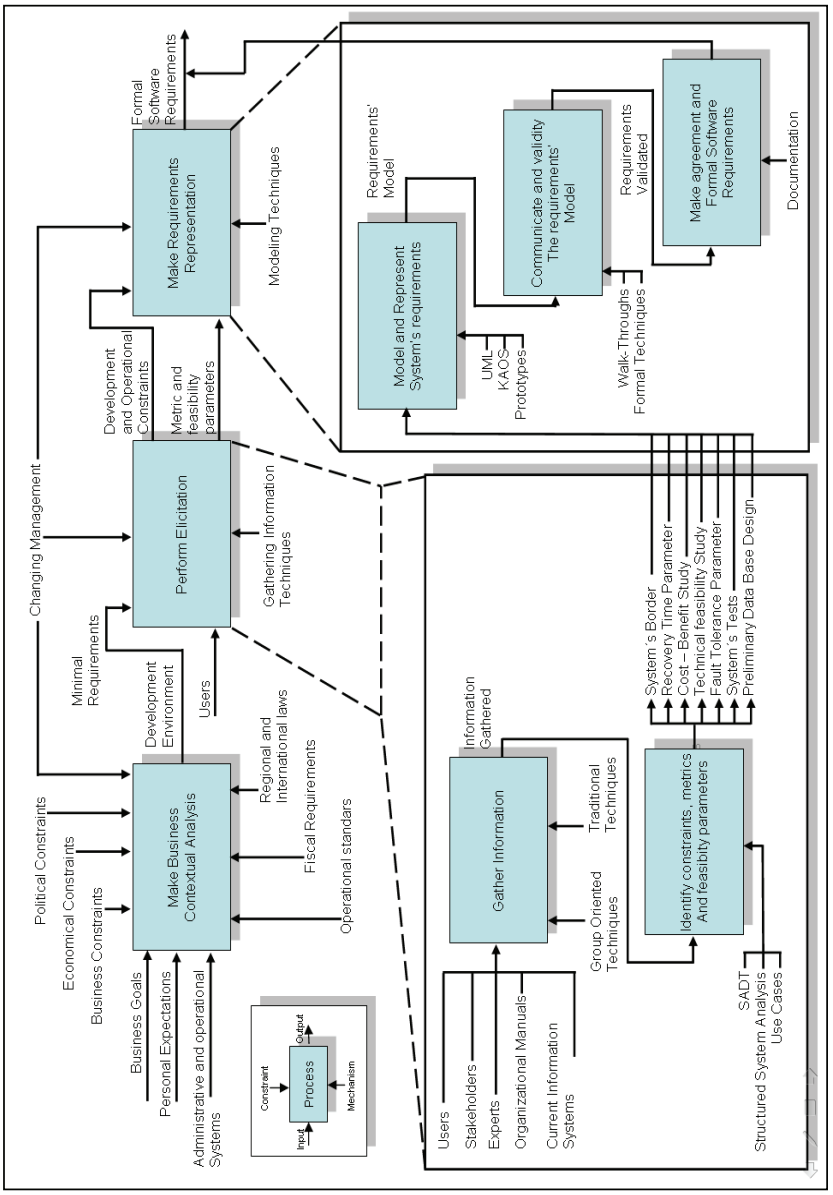
The process proposes three core phases. Phase 1 is "make business contextual analysis," which includes the A.1 social context analysis and A.2. pperational context analysis activities. Phase 2 is "perform elicitation," which includes: A.3 elicitation and A.4 analysis. Phase 3 is "make requirements representation," which includes: A.5 requirements modeling and representation, A.6 requirements communication, and A.7 requirements validation & specification. A "change management" activity is also required, but in this first version of the process, is not considered a full phase. Table 7 shows an initial description each phase and activity for the generic software requirements process.

This generic process was elaborated through the analysis and synthesis of all activities reported in the literature (described in the background section). A main finding that we identified in this study was the relevance played by contextual analysis (and scarcely addressed by most processes analyzed). Contextual analysis considers the environmental influences such as economics, political, business goals, and legal, that could affect the successful

Table 7. Description of phases and activities of the generic SRE process

PHASE		ACTIVITY	DESCRIPTION
MAKE BUSINESS CONTEXTUAL ANALYSIS		A.1 Social Contextual Analysis	To make sense of the social, organizational and user environment where the new software system will be deployed.
		A.2 Operational Contextual Analysis	To identify the operational settings where the new software system will be deployed...
PERFORM ELICITATION		A.3 Elicitation	To identify the explicit (visible) and implicit (non visible) stakeholders' needs that new software system to be deployed must satisfy as well as additional (optional) characteristics.
		A.4 Analysis	To organize and classify needs as well as to evaluate their real (priority and feasibility) need.
MAKE REQUIREMENTS REPRESENTATION		A.5 Modeling and Representation	To translate the needs in graphic, textual or even real system prototypes representations that help to understand the stakeholders' needs...
		A.6 Communication & Negotiation	To let stakeholders know the identified requirements and negotiate the final agreements.
		A.7 Validation and Definitive Specification	To identify potential mistakes, omissions or unfeasible requirements. This activity is conducted simultaneously with A.6. If corrections are required, then will be needed to redo some activities.
A.8	CHANGE MANAGEMENT	To establish procedures control for change management during the full requirements engineering process.	

Figure 1. IDEF0 specification of the generic software requirements engineering process



software system development. Other important aspects are those concerned with training needs and the way that the new software system will affect the current information systems and the current software systems. Then, the contextual analysis activity was added in order to identify such that aspects. A change management activity that is performed after each activity in order to record every change made is also added. With it, a history of the process is available for auditing and continuous improvement issues. All activities are rationally ordered to get the generic process.

In order to complete our generic software engineering process, we identified the actors and their

Table 8. Actors/stakeholders and their roles

Actor	Description	Roles
User	People who will operate the system or will use the information gotten from it	Final User
Expert	People who know the management and operational business processes	Implementation Manager, Process Manager, Tests Manager & Inspection Manager
Customer	People how know the business goals and usually pay for the software product	Process Manager, Customer Interfaces Manager
Business Manager	People who make the expenses control they most know the other actors	Inspection Manager, Meetings Planner
Systems Analyst	Expert personal in Information Systems Analysis and Design	Design Manager, Planning Manager, Project Leader
Development Team	Computer programmers, experts in Graphical User Interfaces	Support Manager
Project Manager	Experts in Requirement Engineering and Information Systems	Planning Manager, Quality Manager, Inspection

roles. As it was established, despite the requirements are attached to a physical system (e.g., the software system), are human beings who define such characteristics. Then, when the requirements are elicited, the most important element are the "actors" (or stakeholders), the information that they own, and their willingness to provide it. An "actor" (or stakeholder) is everyone that affects or is affected for the new system. Actors (as team) know how the system works, what information is needed, where the information comes from, what business goals are, and how to manage projects. They are usually experts doing their work, analyzing information, and making decisions. "Actors" participate in the current organizational systems and processes that an organization wants to support with information software-intensive systems. Actors can rescue a bad project or block a correct one. Actors have enforce and exercise power and politics issues. Table 8 shows the names, description, and roles. These roles were taken from Davis and McHale (2003).

The actors and their level of participation in the activities of the generic software requirements engineering process can be assessed as "participant" or "chairman." "Chairman" means that the actor is responsible for that activity; on the other hand, "participant" means that the actor participates in doing satisfactorily, the activity. It is important to notice that in the activity named "validations & specification," the users (and the remainder of stakeholders) are responsible for doing it, the systems analysts and the development equipment only participate in it; we suggest that because the users have to be sure that their needs were well understood and well identified, the software specification will include them. Table 9 shows such issues.

The theoretical validity of this generic process for requirements engineering was assessed through an evaluation form (available upon request) reported in Mora (2004). According to Mora (2004): *the validation in conceptual researches could be done establishing the extent in which the conceptual model successfully accomplishes with the following criteria: (a) the conceptual model is supported on robust theories and principles; (b) the conceptual model is logically coherent, congruent with the*

Table 9. The level participation of actors/stakeholders in the generic process

Activity		Actors							
Num	Description	User	Expert	Customer	Business Manager	Systems Analyst	Development Team	Project Manager	
A.1	Contextual Analysis					Chairman			Changing Management Influence Area
A.2		Participant	Participant	Participant	Participant		Participant	Participant	
A.3	Elicitation					Chairman			
		Participant	Participant	Participant	Participant		Participant	Participant	
A.4	Analysis					Chairman			
		Participant	Participant	Participant	Participant		Participant	Participant	
A.5	Modeling and Representation					Chairman			
							Participant		
A.6	Communication & Negotiation					Chairman			
		Participant	Participant	Participant	Participant	Participant	Participant	Participant	
A.7	Validation and Specification	Chairman							
		Participant	Participant	Participant	Participant	Participant	Participant	Participant	
A.8	Change Management							Chairman	
		Participant	Participant	Participant	Participant	Participant	Participant	Participant	
		Change Management Influence Area							

Table 10. Theoretical assessment of the generic process

ITEMS	Scores					
	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Average
1. The conceptual model is supported by strong theoretical principles.	3	3	5	5	5	4.2
2. The theoretical principles used are relevant to the topic under study.	NA	5	5	5	5	5
3. The reviewed literature does not have relevant omissions to the related topic.	NA	4	4	4	4	4
4. The conceptual model is logically coherent.	5	5	5	5	5	5
5. The conceptual model is adequate with the purpose for which was designed	4	5	5	5	5	4.8
6. The resultant conceptual model is congruent with the research paradigm used.	5	5	5	5	5	5
7. The conceptual model provides new insights and is not just a duplication of an existing model.	3	4	4	4	5	4
8. The form in which the model is presented is adequate to a scientific study.	4	5	5	5	5	4.8

reality under study, and adequate to the purpose to which is designed; and (c) the conceptual model contributes with something new and it is not a duplication of an existing model. A panel of five experts in software engineering performed the evaluation to determine if the model accomplishes with the three criterions established. The evaluation consists in a questionnaire with eight questions (available upon request). Each item uses a five-point Likert scale. The overall results achieved show that the work is considered theoretically valid for the panel of experts.

FUTURE TRENDS

According to the findings of this research, several surveys on utilization of specific techniques and a related recent study on systems development lifecycles process (SDLCs) (Rodriguez et al., 2008), it can identified that a requirements engineering process is included as a mandatory phase in most SDLCs. Another initial trend is the gradual diminishing of the analysis phase to be incorporated partially to requirements engineering and to design phases in the SDLCs. A final initial trend is that for the case of critical software systems, the security (non functional) set of requirements are mandatory while that for other kinds of systems, this category has been overlapped.

However, the surveys show that not all activities (e.g., elicitation-analysis, modeling-representation, communication-negotiation, validation-specification, and changing management) are followed and not all techniques are used. There is no evidence of a change in this situation. Furthermore, the debate between rigor-discipline versus agile-light oriented SDLCs inhibits a unique trend to deploy a full requirements engineering process. Consequently, a critical challenge for practitioners is the incorporation of such an engineering process as a routine practice. Another challenge identified in this research is the mandatory inclusion of the contextual analysis activity, originally posed by Ross and Schoman (1997) (reference by Brackett, 1990), and extended in this research to enrich the requirement engineering process by considering social, political, legal, and economical issues that surround the external and internal environment. A requirements engineer should know the people's expectations, beliefs and norms, and should also appreciate the fears that the deployment of a new system could generate. Evidences of similar social issues impacts in the development of software have been reported (Curtis et al, 1988).

A final challenge is the updating of the requirements engineering processes to incorporate the management and technical issues that the emergent paradigm of service-oriented software/systems, ICT service management (e.g., based in ITIL), and CRM approaches are demanding. What is useful and what must be generated are core research questions worthy to be pursued. The generic requirements software engineering process posited in this chapter is an initial step towards this challenge.

CONCLUSION

In this conceptual study, a deep review of the literature related to requirements engineering from both a software engineering and information systems perspective is reported, as well as a new "generic process for requirements engineering." The main conclusion to report is that the stage of "software/system requirements engineering," independently of the systems development lifecycle selected for a development team, has been recognized as the most important stage, because the errors introduced during it and discovered in later steps will produce significant cost overruns, delays, and unsatisfied systems' requirements in the project. It is also identified that a "generic process" for the system/software requirements engineering could be used in each software development project. Its adaptation for large, medium, or small projects could also be required. Further research is suggested regarding to the "make business contextual analysis" phase,

which is a primary activity that has been scarcely reported in the software engineering literature. This activity is strongly suggested to be integrated in any "generic process" in order to assist systems designers for acquiring a broader perspective of the business organizational environment in which the software is expected finally to be deployed. Finally, new types of software requirements are identified when critical mission systems are developed. As a main limitation of this study, it must be reported that the "generic process for requirements engineering" elaborated, has not been still empirically tested. However, the results of a theoretical validation from a panel of experts in software engineering suggest a positive and relevant contribution to the disciplines of information systems and software engineering.

REFERENCES

Abbott, R. J. (1986). *An integrated approach to software development*. New York: John Wiley.

Andoh-Baidoo, F., White, E., & Kasper, G. (2004, August). Information systems' cumulative research tradition: A review of research activities and outputs using pro forma abstracts. In *Proceedings of the Tenth Americas Conference on Information Systems* (pp. 4195-4202). New York.

ANSI/IEEE Standard 729-1983. (1983). *IEEE standard glossary of software engineering terminology*. IEEE.

Beckworth, G., & Garner, B. (1994) *An analysis of requirements engineering methods*. School of Computing and Mathematics. Australia: Deakin University.

Brackett, J. (1990). *Software requirements; SEI curriculum module SEI-CM-19-1.2*. Software Engineering Institute: Carnegie Mellon University.

Counelis, J. (2000). *Generic research design in the study of education: a systemic typology. Systems Research, 17*, 51-63.

El-Emam, K., & Madhavji, N. (2005, March). A field study of requirements engineering practices in information systems development. In *Proceedings of the Second IEEE Int. Symp. on Requirements Engineering* (pp. 68-80). York, UK, England.

ESA software engineering standards. ESA Board for Software Standardisation and Control. 1994

ESA PSS-05-03. (1995). *Guide to the software requirements definition phase*. Paris, France: ESA Board for Software Standardisation and Control (BSSC).

Glass, R., Vessey, I., & Ramesh, V. (2002). Research in software engineering: an analysis of the literature. *Information and Software Technology, 44*, 491-506.

González, R. (2005). Developing the requirements discipline: Software vs. systems. *IEEE Software*, *March-April*, 59-61.

Jin, Z., Bell, D., Wilkie, F. G. (1998). Automatically acquiring the requirements fo business information system by using business ontology. In Proceedings of Workshop on Application of Ontologies and Problem Solving Methods (pp. 1-15). Brighton, UK.

Juristo, N., Moreno, A. M., & Silva, A. (2002). Is the European industry moving toward solving requirements engineering problems? *IEEE Software*, *November-December*, 70-77.

Mayer, R. J. (1990). *IDEF0 functional modeling*. College Station, TX: Knowledge Based Systems, Inc.

Mora, M. (2004). *Conceptual research method description*. Aguascalientes, México: UAA. (In Spanish Language).

Mylopoulos, J., Bordiga, A., Jarke, M., & Koubarakis, M. (1999). Telos: representing knowledge about information system. *ACM Trnas. on Information Systems, 8*(4), 325-362.

Neil, C., & Laplante, P. (2003). Requirements engineering: The state of the practice. *IEEE Software, November-December*, 40-45.

Nikula, U., Fajaniemi, J. & Kalviainen, H. (2000) Management view on current requirements engineering practices in small and medium enterprises. In *Proceedings of the Fifth Australian Workshop on Requirements Engineering* (pp. 81-89). Queensland University of Technology, Brisbane, Australia, Faculty of Information Technology, University of Sydney (UTS).

Nuseibeh, B., & Easterbrook, S. (2000). *Requirements engineering: A roadmap*. ACM Digital Library, 1/58113-253-0/00-6.

Rodríguez, L. C. (2006). *Comparative study on systems development life-cycles process*. Aguascalientes, Mexico: UAA.

Ross, D.T., & Schoman, K.E., Jr. (1997). Structured analysis for requirements definition. *IEEE Trans. Software Eng., SE-3*, (1), 6-15.

Sage, A., & Armstrong, J. (2000). *Introduction to systems engineering*. New York: Wiley.

Sawyer, P., & Kotonya, G. (2001). *Chapter 2 software requirements*. SWEBOK, IEEE-Trial Version 1.00-May.

Sommerville, I. (2005). *Software engineering*. Addison Wesley.

Sumano, M. (1999). *Stated of the art of software requirements analysis* (Technical Report No. 33). Mexico: Computer Sciences Research Center, National Politechnical Institute. (In Spanish Language).

SWEBOK. (2004). *Guide to the software engineering body of knowledge*. A project of the IEEE Computer Society Professional Practices Committee.

Viennau, R. (1993). A review of formal methods. *In A Review of Formal Methods, Kaman Science Corp.* (3-15).

Zave, P. (1997). Classification of research efforts in requirements engineering. *ACM Computing Surveys, 29*(4), 315-321.

KEY TERMS

Changing Management: Control of every change proposal made during the software specification process.

Contextual Analysis: Analysis of the social, technical, and operational environment where the system software will be developed and operated.

Generic Process: Universal process that can be used for every software development project.

Operational Context Analysis: Analysis of the operational environment where the system software will be developed and operated.

Requirements Elicitation: Process of discovering the requirements for a system by communication with customers, system users, and others who have a stake in the system development.

Requirement Engineering: Discipline that establishes the services that a software system must provide, and constrains that must satisfy.

Social Context Analysis: Analysis of the social environment where the system software will be developed and operated.