Analysis and Need of Requirements Engineering

Ranjeet Kaur
Lecturer in Information Technology
GIMET,Amritsar

Tajinder Singh
Lecturer in Information Technology
GIMET,Amritsar

ABSTRACT

This paper presents outline of the field of software systems requirements engineering (RE). It describes the main areas of RE practice, and highlights some key open Research issues for the future and what is RE all about? When is it needed? What kinds of activities are involved in doing RE? Requirements engineering applies to the development of all software-intensive systems, but not necessarily to the development of all software, as we shall see. There are a huge range of different kinds of software-intensive system, and the practice of RE varies across this range. Our aim throughout this paper is to explore both what is common and what varies across these different types of system. The key techniques used in requirements engineering for dealing with complexity.

Keywords: RE-Requirement Engineering, Software-intensive system, System

1. INTRODUCTION

The primary appraise of success of a software system is the degree to which it meets the purpose for which it was intended. Generally speaking, software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is agreeable to analysis, communication, and subsequent implementation. There are a number of intrinsic difficulties in this process. Stakeholders (including paying customers, users and developers) may be several and distributed. Their goals may vary and conflict, depending on their perspectives of the environment in which they work and the tasks they wish to accomplish. Their goals may not be explicit or may be difficult to communicative, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control. In this paper we present an overview of current research in RE, presented in terms of the main activities that constitute the field. While these activities are described independently and in a particular order, in practice, they are actually interleaved, iterative, and may span the entire software systems development life cycle.

1.1 The Core RE Activities are:

1. Eliciting requirements,
2. Modeling and analyzing requirements,
3. Communicating requirements,
4. Agreeing requirements, and
5. Evolving requirements.

2. BASIC OF RE:

"**Requirements Engineering (RE)** is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies."

This definition is attractive for a number of reasons. First, it highlights the importance of "real-world goals" that motivate the development of a software system. These represent the 'why' as well as the 'what' of a system. Second, it refers to "precise specifications". These provide the basis for analyzing requirements, validating that they are indeed what stakeholders want, defining what designers have to build, and verifying that they have done so correctly upon delivery. Finally, the definition refers to specifications' "evolution over time and across software families", emphasizing the reality of a changing world and the need to reuse partial specifications, as engineers often do in other branches of engineering. It has been argued that requirements engineering is a misnomer. Typical textbook definitions of engineering refer to the creation of cost-effective solutions to practical problems by applying scientific knowledge. Therefore, the use of the term engineering in RE serves as a reminder that RE is an important part of an engineering process, being the part concerned with anchoring development activities to a real-world problem, so that the appropriateness and cost-effectiveness of the solution can then be analyzed. It also refers to the idea that specifications themselves need to be engineered, and RE represents a series of engineering decisions that lead from recognition of a problem to be solved to a detailed specification of that problem .The tools and techniques used in RE draw upon a variety of disciplines, and the requirements engineer may be expected to master skills from a number of different disciplines. In the context of software development, computer science plays a particularly important role. Theoretical computer science provides the framework to assess the feasibility of requirements, while practical computer science provides the tools by which software solutions are developed. Although software engineering still lacks a mature science of software behavior on which to draw, requirements engineers need such a science in order to understand how to specify the required behavior of software. Since software is a formal description, analysis of its behavior is amenable to formal reasoning. Logic provides a vehicle for performing such analysis [1]. A further advantage of specification languages grounded in logic is that they are potentially amenable to automated reasoning and analysis. In the systems engineering context, an understanding and application of systems theory and practice is also relevant to RE. This includes work on characterizing systems, identifying their boundaries and managing their development life cycle. RE also encompasses work on systems analysis, traditionally found in the information systems world . The context in which RE takes place is usually a human activity system, and the

problem owners are people. Engagement in an RE process presupposes that some new computer-based system could be useful, but such a system will change the activities that it supports. Therefore, RE needs to be sensitive to how people perceive and understand the world around them, how they interact, and how the sociology of the workplace affects their actions. RE draws on the cognitive and social sciences to provide both theoretical grounding and practical techniques for eliciting and modeling requirements:

2.1 Cognitive Psychology provides an understanding of the difficulties people may have in describing their needs. For example, problem domain experts often have large amounts of tacit knowledge that is not amenable to introspection; hence their answers to questions posed by requirements analysts may not match their behavior. Also, the requirements engineer may need to model users' understanding of software user interfaces, rather than relying solely on implementers' preferences.

2.2 Anthropology provides a ethodological approach to observing human activities that helps to develop a richer understanding of how computer systems may help or hinder those activities. For example, the techniques of ethno methodology have been applied in RE to develop observational techniques for analyzing collaborative work and team interaction.

2.3 Sociology provides an understanding of the political and cultural changes caused by computerization. Introduction of a new computer system changes the nature of the work carried out within an organization, may affect the structure and communication paths within that organization, and may even change the original needs that it was built to satisfy. A requirements gathering exercise can therefore become politicized.

3. DEALING WITH COMPLEXITY

Requirements Engineering offers a number of techniques for dealing with complexity of purpose, which are built into the various techniques described in this book. Of these, three general principles are so useful that we will briefly introduce them here: abstraction, decomposition and projection:

3.1. Abstraction involves ignoring the details so that we can see the big picture. When we take some set of human-computer activities and describe them as a system, we are using an abstraction. When we take two different actions and describe them as instances of the same general activity, we are using an abstraction.

3.2 Decomposition involves breaking a set of phenomena into parts, so that we can study them independently. Such decompositions are never perfect, because of the coupling between the parts, but a good decomposition still offers us insights into how things work.

3.3 Projection involves adopting a particular view or perspective, and describing only the aspects that are relevant to that perspective. Unlike decomposition, the perspectives are not intended to be independent in any way.

These ideas are so useful that we use them all the time, often without realizing it. Requirements analysts use them in a particular way to understand problem situations, and to identify parts of a problem that can be solved using software. Systematic use of decomposition, abstraction and projection allows us to deal with complexity by making problems simpler, and mapping them on to existing solution components.

4. ELICTING REQURIMENTS

The elicitation of requirements is perhaps the activity most often regarded as the first step in the RE process. The term "elicitation" is preferred to "capture", to avoid the suggestion that requirements are out there to be collected simply by asking the right questions. Information gathered during requirements elicitation often has to be interpreted, analyzed, modeled and validated before the requirements engineer can feel confident that a complete enough set of requirements of a system have been collected. Therefore, requirements elicitation is closely related to other RE activities – to a great extent, the elicitation technique used is driven by the choice of modeling scheme, and vice versa: many modeling schemes imply the use of particular kinds of elicitation techniques.

4.1 Requirements to Elicit

One of the most important goals of elicitation is to find out what problem needs to be solved, and hence identify system boundaries. These boundaries define, at a high level, where the final delivered system will fit into the current operational environment. Identifying and agreeing a system's boundaries affects all subsequent elicitation efforts. The identification of stakeholders and user classes, of goals and tasks, and of scenarios and use cases all depend on how the boundaries are chosen. Identifying stakeholders – individuals or organizations who stand to gain or lose from the success or failure of a system – is also critical. Stakeholders include customers or clients, developers, and users. Goals denote the objectives a system must meet. Eliciting high-level goals early in the development process is crucial. However, goal-oriented requirements elicitation [8] is an activity that continues as development proceeds, as high-level goals are refined into lower level goals. Eliciting goals focuses the requirements engineer on the problem domain and the needs of the stakeholders, rather than on possible solutions to those problems. It is often the case that users find it difficult to articulate their requirements. To this end, a requirements engineer can resort to eliciting information about the tasks users currently perform and those that they might want to perform. These tasks can often be represented in use cases that can be used to describe the outwardly visible requirements of systems.

4.2 Elicitation Techniques

The choice of elicitation technique depends on the time and resources available to the requirements engineer, and of course, the kind of information that needs to be elicited. We distinguish a number of classes of elicitation technique:

4.2.1 Traditional Techniques include a broad class of generic data gathering techniques. These include the use of questionnaires and surveys, interviews, and analysis of existing documentation such as organizational charts, process models or standards, and user or other manuals of existing systems.

4.2.2 Group Elicitation Tehniques aim to foster stakeholder agreement and buy-in, while exploiting team dynamics to elicit a richer understanding of needs. They include brainstorming and focus groups, as well as RAD/JAD workshops.

4.2.3 Prototyping has been used for elicitation where there is a great deal of uncertainty about the requirements, or where early feedback from stakeholders is needed. Prototyping can also be readily combined with other techniques, for instance by using a prototype to provoke discussion in a group elicitation technique, or as the basis for a questionnaire or think-aloud protocol.

4.2.4 Model-Driven Techniques provide a specific model of the type of information to be gathered, and use this model to drive the elicitation process. These include goal-based methods, and scenario-based methods.

4.2.5 Cognitive Techniques include a series of techniques originally developed for knowledge acquisition for knowledge-based systems. Such techniques include protocol analysis (in which an expert thinks aloud while performing a task, to provide the observer with insights into the cognitive processes used to perform the task), laddering (using probes to elicit structure and content of stakeholder knowledge), card sorting, and repertory grids.

4.2.6 Contextual Techniques emerged in the 1990's as an alternative to both traditional and cognitive techniques. These include the use of ethnographic techniques such as participant observation. They also include ethnomethodogy and conversation analysis, both of which apply fine-grained analysis to identify patterns in conversation and interaction. To some extent, there is a fundamental methodological disagreement between the proponents of contextual techniques on the one hand, and the traditional and cognitive techniques on the other. Contextual approaches are based on the premise that local context is vital for understanding social and organizational behavior, and the observer must be immersed in this local context in order to experience how participants create their own social structures.

4.3 The Elicitation Process

With a overabundance of elicitation techniques available to the requirements engineer, some guidance on their use is needed. Methods provide one way of delivering such guidance. Each method itself has its strengths and weaknesses, and is normally best suited for use in particular application domains. For example, the Inquiry Cycle [14] and CREWS [12] provide alternative methods for eliciting requirements using use cases and scenarios. Of course, in some circumstances a full-blown method may be neither required nor necessary. Instead, the requirements engineer needs simply to select the appropriate technique or techniques most suitable for the elicitation process in hand. In such situations, technique-selection guidance is more appropriate than a rigid method.

5. MODELLING AND ANALYSING REQURIMENTS

Modeling – the construction of abstract descriptions that are amenable to interpretation – is a fundamental activity in RE. So much so that a number of RE textbooks focus almost entirely on modeling methods and their associated analysis techniques. Models can be used to represent a whole range of products of the RE process. Moreover, many modeling approaches are used as elicitation tools, where the modeling notation and partial models produced are used as drivers to prompt further information gathering.

The key question to ask for any modeling approach is "what is it good for?", and the answer should always be in terms of the kind of analysis and reasoning it offers. We suggest below some general categories of RE modeling approaches, and give some example techniques under each category. We then suggest some analysis techniques that can be used to generate useful information from the models produced.

5.1 Enterprise Modeling

The context of most RE activities and software systems is an organization in which development takes place or in which a system will operate. Enterprise modeling and analysis deals with understanding an organization's structure; the business rules that affect its operation; the goals, tasks and responsibilities of its constituent members; and the data that it needs, generates and manipulates. Enterprise modeling is often used to capture the purpose of a system, by describing the behavior of the organization in which that system will operate. This behavior can be expressed in terms of organizational objectives or goals and associated tasks and resources. Others prefer to model an enterprise in terms of its business rules, workflows and the services that it will provide. Modelling goals is particularly useful in RE. High-level business goals can be refined repeatedly as part of the elicitation process, leading to requirements that can then be operationalised [8].

5.2 Data Modeling

Large computer-based systems, especially information systems use and generate large volumes of information. This information needs to be understood, manipulated and managed. Careful decisions need to be made about what information the system will need to represent, and how the information held by the system corresponds to the real world phenomena being represented. Data modeling provides the opportunity to address these issues in RE. Traditionally, Entity-Relationship-Attribute (ERA) modeling is used for this type of modeling and analysis. However, object-oriented modeling, using class and object hierarchies, are increasingly supplanting ERA techniques.

5.3 Behavioral Modeling

Modeling requirements often involves modeling the dynamic or functional behavior of stakeholders and systems, both existing and required. The distinction between modeling an existing system, and modeling a future system is an important one, and is often blurred by the use of the same modeling techniques for both. Early structured analysis methods suggested that one should start by modeling how the work is currently carried out (the current physical system), analyze this to determine the essential functionality (the current logical system), and finally build of model of how the new system ought to operate (the new logical system). Explicitly constructing all three models may be overkill, but it is nevertheless useful to distinguish which of these is being modeled. A wide range of modeling methods are available, from structured to object-oriented methods, and from soft to formal methods. These methods provide different levels of precision and are amenable to different kinds of analysis. Formal methods can be difficult to construct, but are also amenable to automated analysis. On the other hand, soft methods provide rich representations that non-technical stakeholders find appealing, but are often difficult to check automatically.

5.4 Domain Modeling

A significant proportion of the RE process is about developing domain descriptions. A model of the domain provides an abstract description of the world in which an envisioned system will operate. Building explicit domain models provides two key advantages: they permit detailed reasoning about (and therefore validation of) what is assumed about the domain, and they provide opportunities for requirements reuse within a domain. Domain-specific models have also been shown to be essential for building automated tools, because they permit tractable reasoning over a closed world model of the system interacting with its environment.

5.5 Modeling Non-Functional requirements (NFRs)

Non-functional requirements (also known as quality requirements) are generally more difficult to express in a measurable way, making them more difficult to analyze. In particular, NFRs tend to be properties of a system as a whole, and hence cannot be verified for individual components. Recent work by both researchers [7] and practitioners has investigated how to model NFRs and to express them in a form that is measurable or testable. There also is a growing body of research concerned with particular kinds of NFRs, such as safety security [6], reliability [9], and usability.

5.6 Analyzing Requirements Models

A primary benefit of modeling requirements is the opportunity this provides for analyzing them. Analysis techniques that have been investigated in RE include requirements animation, automated reasoning consistency checking (e.g., model checking), and a variety of techniques for validation and verification (V&V).

5.7 Communicating Requirements

RE is not only a process of discovering and specifying requirements; it is also a process of facilitating effective communication of these requirements among different stakeholders. The way in which equirements are documented plays an important role in ensuring that they can be read, analyzed, (re-)written, and validated. The focus of requirements documentation research is often on specification languages and notations, with a variety of formal, semi-formal and informal languages suggested for this purpose. One attempt to achieve readability has been the development of a variety of documentation standards that provide guidelines for structuring requirements documents.

5.8 Agreeing Requirements

As requirements are elicited and modeled, maintaining agreement with all stakeholders can be a problem, especially where stakeholders have divergent goals. Recall that *validation* is the process of establishing that the requirements and models elicited provide an accurate account of stakeholder requirements. Explicitly describing the requirements is a necessary precondition not only for validating requirements, but also for resolving conflicts between stakeholders. Techniques such as inspection and formal analysis tend to concentrate on the coherence of the requirements descriptions: are they consistent, and are they structurally complete? The formal method SCR [10] illustrates this approach. The SCR tool provides automated checking that the formal model is syntactically consistent and complete. In contrast, techniques such as prototyping, specification animation, and the use of scenarios are geared towards testing a correspondence with the real world problem. For example, have all the aspects of the problem that the stakeholders regard as important been covered? Requirements validation is difficult for two reasons. The first reason is philosophical in nature, and concerns the question of truth and what is knowable. The second reason is social, and concerns the difficulty of reaching agreement among different stakeholders with conflicting goals. We will briefly examine each of these in turn. We can compare the problem of validating requirements with the problem of validating scientific knowledge. Many requirements engineers adopt a logical positivist approach – essentially the belief that there is an objective world that can be modeled by building a consistent body of knowledge grounded in empirical observation. In RE, this view says that the requirements describe some objective problem that exists in the world, and that validation is the task of making sufficient empirical observations to check that this problem has been captured correctly.

5.9 Evolving Requirements

Successful software systems always evolve as the environment in which these systems operate changes and stakeholder requirements change. Therefore *managing change* is a fundamental activity in RE [4]. Changes to requirements documentation need to be managed. Minimally, this involves providing techniques and tools for configuration management and version control, and exploiting tractability links to monitor and control the impact of changes in different parts of the documentation. Typical changes to requirements specifications include adding or deleting requirements, and fixing errors. Requirements are added in response to changing stakeholder needs, or because they were missed in the initial analysis. Requirements are deleted usually only during development, to forestall cost and schedule overruns, a practice known as *requirements scrubbing* [3].In any case, managing inconsistency in requirements specifications as they evolve is a major challenge. Inconsistencies arise both as a result of mistakes, and because of conflicts between requirements. Each inconsistency implies that some action is needed, to identify the cause and seek a resolution. While tractability links help to scope the possible impact of change, they do not support automated reasoning about change, because the links carry little semantic information. One attempt to address this problem is the Viewpoints framework, in which consistency relationships between chunks (‘viewpoints’) of a specification are expressed operationally, so that automated support for propagation of change becomes possible. Managing changing requirements is not only a process of managing documentation, it is also a process of recognizing change through continued requirements elicitation, reevaluation of risk, and evaluation of systems in their operational environment. In software engineering, it has been demonstrated that focusing change on program code leads to a loss of structure and maintainability [2]. Thus, each proposed change needs to be evaluated in terms of existing requirements and architecture so that the trade-off between the cost and benefit of making a change can be assessed.

Finally, the development of software system *product families* has become an increasingly important form of development activity. For this purpose, there is a need to develop a range of software products that share similar requirements and architectural characteristics, yet differ in certain key requirements. The process of identifying *core requirements* in order to develop architectures that are (a) stable in the

presence of change, and (b) flexible enough to be customized and adapted to changing requirements, is one of the key research issues in software engineering.

5.10 Integrated Requirements Engineering

RE is a multi-disciplinary activity, deploying a variety of techniques and tools at different stages of development and for different kinds of application domains. Methods provide a systematic approach to combining different techniques and notations, and *method engineering* [5] plays an important role in designing the RE process to be deployed for a particular problem or domain. Methods provide heuristics and guidelines for the requirements engineer to deploy the appropriate notation or modeling technique at different stages of the process. A variety of approaches have been suggested to manage and integrate different RE activities and products. Jackson, for example, uses problem frames to structure different kinds of elementary and composite problems. His argument is that identifying well-understood problems offers the possibility of selecting corresponding, appropriate, well-understood, solutions. An alternative approach to organizing, selecting and tailoring multiple methods is through the use of multiple perspectives or views of requirements. This approach can facilitate requirements partitioning and subsequent modeling and analysis. For example, a viewpoint can be treated as an encapsulation of an individual technique, with a defined notation, a set of actions that can be performed on that notation, and a set of rules for consistency relationships with other viewpoints. In this way, the design and integration of multiple methods can be supported as a process of creating and tailoring viewpoint templates. Finally, to enable effective management of an integrated RE process, automated tool support is essential. Requirements management tools, such as DOORS, Requisite Pro, Cradle, and others, provide capabilities for documenting requirements, managing their change, and integrating them in different ways depending on project needs.

6. A REQUIREMENTS ENGINEERING DRAFT

This paper has set out a draft, and we feel that no draft is complete without a big arrow labeled you are here"1. By way of providing such a marker, we will summarize the important developments in RE during the last decade, and give our predictions about what will be important in RE research for the coming decade. The 1990's saw several important and radical shifts in the understanding of RE. By the early 1990's, RE had emerged as a field of study in its own right, as witnessed by the emergence of two series of international meetings – the IEEE sponsored conference and symposium, held in alternating years – and the establishment of an international journal published by Springer. By the late 1990's, the field had grown enough to support a large number of additional smaller meetings and workshops in various countries. During this period, we can discern the emergence of three radical new ideas that challenged and overturned the orthodox views of RE. These three ideas are closely interconnected:

6.1 The idea that modeling and analysis cannot be performed adequately in isolation from the organizational and social context in which any new system will have to operate. This view emphasized the use of conceptualized enquiry techniques, including ethno methodology and participant observation.

6.2 The notion that RE should *not* focus on specifying the functionality of a new system, but instead should concentrate on modeling indicative and *optative* properties of the *environment*. Only by describing the environment, and expressing what the new system must achieve in that environment, we can capture the system's purpose, and reason about whether a given design will meet that purpose.

6.3 The idea that the attempt to build consistent and complete requirements models is useless, and that RE has to take seriously the need to analyze and resolve conflicting requirements, to support stakeholder negotiation, and to reason with models that contain inconsistencies. Having identified these trends from the past decade, we now turn our attention to the future. We believe the following represent major challenges for RE in the years ahead:

6.3.1 Development of new techniques for formally modeling and analyzing properties of the environment, as opposed to the 1 Sadly, this is an infeasible requirement for most portable road maps! 2 *Indicative* descriptions express things that are currently true (and will be true irrespective of the introduction of a new system), while *optative* descriptions express the things that we wish the new system to make true. behavior of the software. Such techniques must take into account the need to deal with inconsistent, incomplete, and evolving models. We expect such approaches will better support areas where RE has been weak in the past, including the specification of the expectations that a software component has of its environment. This facilitates migration of software components to different software and hardware environments, and the adaptation of products into product families.

6.3.2 Bridging the gap between requirements elicitation approaches based on contextual enquiry and more formal specification and analysis techniques. Contextual approaches, such as those based on ethnographic techniques, provide a rich understanding of the organizational context for a new software system, but do not map well onto existing techniques for formally modeling the current and desired properties of problem domains. This includes the incorporation of a wider variety of media, such as video and audio, into behavioral modeling techniques.

6.3.3 Richer models for capturing and analyzing non-functional requirements. These are also known as the "ilities" and have defied a clear characterization for decades.

6.3.4. Better understanding of the impact of software architectural choices on the prioritization and evolution of requirements. While work in software architectures has concentrated on how to express software architectures and reason about their behavioral properties, there is still an open question about how to analyze what impact a particular architectural choice has on the ability to satisfy current and future requirements, and variations in requirements across a product family.

6.3.5 Reuse of requirements models. We expect that in many domains of application, we will see the development of reference models for specifying requirements, so that the effort of developing requirements models from scratch is reduced. This will help move many software projects from being creative design to being normal design, and will facilitate the selection of commercial off-the-shelf software [13].

6.3.6 Multidisciplinary training for requirements practitioners. In this paper, we have used the term "requirements engineer" to refer to any development participant who applies the techniques described in the paper to elicit, specify, and analyze requirements. While many organizations do not even employ such a person, the skills that such a person or group should possess is a matter of critical importance. The requirements engineer must possess both the social skills to interact with a variety of stakeholders, including potentially non-technical customers, and the technical skills to interact with systems designers and developers. Many delivered systems do not meet their customers' requirements due, at least partly, to ineffective RE. RE is often treated as a time-consuming, bureaucratic and contractual process. This attitude is changing as RE is increasingly recognized as a critically important activity in any systems engineering process. The novelty of many software applications, the speed with which they need to be developed, and the degree to which they are expected to change, all play a role in determining how the systems development process should be conducted. The demand for better, faster, and more usable software systems will continue, and RE will therefore continue to evolve in order to deal with different development scenarios. We believe that effective RE will continue to play a key role in determining the success or failure of projects, and in determining the quality of systems that are delivered.

7. REFERENCES

[1] Abramsky, S., Gabbay, D. & Maibaum, T. (Ed.). (1992). *Handbook of Logic in Computer Science Vol 1: Background: Mathematical Structures*. Clarendon Press.

[2] Bennett, K. H. & Rajlich, V. T. (2000). Software Maintenance and Evolution.

[3] Boehm, B. (1991). Software Risk Management: Principles and Practices. *IEEE Software*, 8(1): 32-41.

[4] Bohner, S. A. & Arnold, R. S. (Ed.). (1996). *Software Change Impact Analysis*. IEEE Computer Society Press.

[5] Brinkkemper, S. & Joosten, S. (1996). Editorial: Method Engineering and Meta-modelling. *Information and Software Technology*, 38(4): 259.

[6] Chung, L. (1993). Dealing with Security Requirements During the Development of Information Systems. *5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*, Paris, France, 1993, pp. 234-251.

[7] Chung, L., Nixon, B., Yu, E. & Mylopoulos, J. (2000). *Non- Functional Requirements in Software Engineering*. Boston: Kluwer Academic Publishers.

[8] Dardenne, A., Lamsweerde, A. v. & Fickas, S. (1993). Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20:3-50.

[9] del Gobbo, D., Napolitano, M., Callahan, J. & Cukic, B. (1998.). Experience in Developing System Requirements Specification for a Sensor Failure Detection and Identification Scheme. *3rd High Assurance Systems Engineering Symposium*, Washington, DC, USA, 13-14 November 1998.

[10] Heitmeyer, C. L., Jeffords, R. D. & Labaw, B. G. (1996). Automated Consistency Checking of Requirements Specifications. *IEEE Transactions on Software Engineering and Methodology*, 5(3): 231-261.

[11] Maiden, N. (1998). CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, 5(4):419-446.

[12] Maiden, N. & Rugg, G. (1996). ACRE: Selecting Methods For Requirements Acquisition. *Software Engineering Journal*, 11(3):183-192.

[13] Maiden, N. A. M. & Ncube, C. (1998). Acquiring Requirements for Commercial Off-The-Shelf Package Selection. *IEEE Software*, 15(2):46-56.

[14] Potts, C., Takahashi, K. & Anton, A. (1993). Inquiry-based requirements Analysis. *IEEE Software*, 11(2): 21-32.