Requirements Engineering: An Overview

Dr. K. Sunil Manohar Reddy

Associate Professor, Dept. Of Computer Science and Engineering, Matrusri Engineering College, Hyderabad, Telangana, India

sunil186@gmail.com

***Abstract*—** This paper provides an insight into software systems requirements engineering. It clearly explains the main areas of requirements engineering practice, and specifies some key open research issues for the future.

Keywords*—Requirements Engineering, Elicit, Validate, Analyze, Prototyping.*

I. INTRODUCTION

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly 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 amenable to analysis, communication, and subsequent implementation. There are a number of inherent difficulties in this process. Stakeholders (including paying customers, users and developers) may be numerous 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 articulate, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control.

This paper gives 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.

II. FOUNDATIONS

Before discussing RE activities in more detail, it is worth examining the role of RE in software and systems engineering, and the many disciplines upon which it draws. Zave [83] provides one of the clearest definitions of RE:

"Requirements engineering is the branch of software engineering concerned with the real world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

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 *analysing* 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", emphasising 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 [74]. 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.

Note that the focus of Zave's definition is on software engineering. In reality, software cannot function in isolation from the system in which it is embedded, and hence RE has to encompass a systems level view. We therefore prefer to characterise RE as a branch of *systems engineering* [76], whose ultimate goal is to deliver some systems behavior to its stakeholders. The special consideration that *software systems requirements engineering* has received is largely due to the abstract and invisible nature of software, and the vast range and variety of problems that admit to software solutions.

Whether viewed at the systems level or the software level, RE is a multi-disciplinary, human-centered process. 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]. In RE, logic can be used to improve the rigour of the analysis performed, and to make the reasoning steps explicit. Formal description techniques have received considerable attention in RE research, but have not yet been widely adopted into RE practice. Since RE must span the gap between the informal world of stakeholder needs, and the formal world of software behavior, the key question over the use of formal methods is not *whether* to formalise, but *when* to formalise [60]. Different logics may be used to express different aspects of a required system. 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 [76]. This includes work on characterising systems, identifying their boundaries and managing their development life cycle [12]. RE encompasses work on systems analysis, traditionally found in the information systems world [68].

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:

- *Cognitive psychology* provides an understanding of the difficulties people may have in describing their needs [62]. 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.
- *Anthropology* provides a methodological approach to observing human activities that helps to develop a richer understanding of how computer systems may help or hinder those activities [29]. For example, the techniques of ethno methodology [30] have been applied in RE to develop observational techniques for analysing collaborative work and team interaction.
- *Sociology* provides an understanding of the political and cultural changes caused by computerisation. 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 [46]. A requirements gathering exercise can therefore become politicized. Approaches to RE that address this issue include the "Scandanavian" approach, which aims to involve in the requirements definition process those most affected by the outcomes [36].
- *Linguistics* is important because RE is largely about communication. Linguistic analyses have changed the way in which the English language is used in specifications, for instance to avoid ambiguity and to improve understandability. Tools from linguistics can also be used in requirements elicitation, for instance to analyse communication patterns within an organization [11].

Finally, there is an important philosophical element in RE. RE is concerned with interpreting and understanding stakeholder terminology, concepts, viewpoints and goals. Hence, RE must concern itself with an understanding of beliefs of stakeholders (*epistemology*), the question of what is observable in the world (*phenomenology*), and the question of what can be agreed on as objectively true (*ontology*). Such issues become important whenever one wishes to talk about validating requirements, especially where stakeholders may have divergent goals and incompatible belief systems. They also become important when selecting a modeling technique, because the choice of technique affects the set of phenomena that can be modeled, and may even restrict what a requirements engineer is capable of observing.

III. CONTEXT AND GROUNDWORK

RE is often regarded as a front-end activity in the software systems development process. This is generally true, although it is usually also the case that requirements change during development and evolve after a system has been in operation for some time. Therefore, RE plays an important role in the management of change in software development. Nevertheless, the bulk of the effort of RE does occur early in the lifetime of a project, motivated by the evidence that requirements errors, such as misunderstood or omitted requirements, are more expensive to fix later in project lifecycles [8; 56].

Before a project can be started, some preparation is needed. Finkelstein [24] categorises such preparation as *context* and *groundwork*. In the past, it was often the case that RE methods assumed that RE was performed for a specific customer, who could sign off a requirements specification. However, RE is actually performed in a variety of contexts, including market-driven product development and development for a specific customer with the eventual intention of developing a broader market. The type of product will also affect the choice of method: RE for information systems is very different from RE for embedded control systems, which is different again from RE for generic services such as networking and operating systems.

For groundwork, some assessment of a project's feasibility and associated risks needs to be undertaken, and RE plays a crucial role in making such an assessment. It is often possible to estimate project costs, schedules and technical feasibility from precise specifications of requirements. It is also important that conflicts between high-level goals of an envisioned system surface early, in order to establish a system's concept of operation and boundaries. Of course, risk should be re-evaluated regularly throughout the development lifetime of a system [58], since changes in the environment can change the associated development risks.

Groundwork also includes the identification of a suitable process for RE, and the selection of methods and techniques for the various RE activities. We use the term *process* here to denote an instance of a process model, which is an abstract description of how to conduct a collection of activities, describing the behaviour of one or more agents and their management of resources. A *technique* prescribes how to perform one particular activity - and, if necessary, how to describe the product of that activity in a particular notation. A *method* provides a prescription to perform a collection of activities, focusing on how a related set of techniques can be integrated, and providing guidance on their use.

IV. ELICITING REQUIREMENTS

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 [29]. 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 (who pay for the system), developers (who design, construct and maintain the system), and users (who interact with the system to get their work done). For interactive systems, users play a central role in the elicitation process, as usability can only be defined in terms of the target user population. Users themselves are not homogeneous, and part of the elicitation process is to identify the needs of different user classes, such as novice users, expert users, occasional users, disabled users, and so on [73].

Goals denote the objectives a system must meet. Eliciting high level goals early in the development process is crucial. However, goal-oriented requirements elicitation [15] is an activity that continues as development proceeds, as high- level goals (such as business goals) are refined into lower- level goals (such as technical goals that are eventually operationalised in a system). 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 [42]. These tasks can often be represented in *use cases* that can be used to describe the outwardly visible requirements of systems [72]. More specifically, the requirements engineer may choose a particular path through a use case, a *scenario*, in order to better understand some aspect of using a system [41].

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:

- *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, and user manuals of existing systems.
- *Group elicitation techniques* 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 (using consensus-building workshops with an unbiased facilitator) [52].
- *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 [17]. 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.
- *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, such as KAOS [79] and I* [14], and scenario-based methods such as CREWS [51].
- *Cognitive techniques* include a series of techniques originally developed for knowledge acquisition for knowledge-based systems [75]. 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* (asking stakeholders to sort cards in groups, each of which has name of some domain entity), *repertory grids* (constructing an attribute matrix for entities, by asking stakeholders for attributes applicable to entities and values for cells in each entity).
- *Contextual techniques* emerged in the 1990's as an alternative to both traditional and cognitive techniques [30]. These include the use of ethnographic techniques such as participant observation. They also include ethno methodogy and conversation analysis, both of which apply

fine grained analysis to identify patterns in conversation and interaction [80].

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. The emergence of contextual techniques in RE in the early 1990's paralleled their introduction as part of a revolution in cognitive science and human-computer interaction, where they reflected a blistering attack on the attempt to build disembodied models of cognition [57]. In their extreme forms, the two sides are incompatible: traditional and cognitive approaches are based on the use of abstracted models that are independent of context, whilst the contextualists insist that context is paramount, and completely resist any attempt to build generalisable models of the phenomena they observe. However, it does seem that the advantages of these alternative approaches are complementary, and recent work has focused on the question of whether integration is possible [63; 80].

4.3 The Elicitation Process

With a plethora 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 [64] and CREWS [51] 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 [52].

V. MODELING AND ANALYZING REQUIREMENTS

Modeling – the construction of abstract descriptions that are amenable to interpretation – is a fundamental activity in RE. 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 [47]. This behavior can be expressed in terms of organizational objectives or goals and associated tasks and resources [82]. Others prefer to model an enterprise in terms of its business rules, workflows and the services that it will provide [33].

Modeling 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 [15].

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), analyse 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 [71]. On the other hand, soft methods provide *rich* representations [63] that non-technical stakeholders find appealing, but are often difficult to check automatically.

The deliverable from the requirement's phase is the business Use Case Model. This model consists of UML Use Case diagrams and essential use cases. The intent of these models is to describe the functions of the business in a technology-independent way.

5.4 Domain Modeling

A significant proportion of the RE process is about developing *domain descriptions* [40]. 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 analyse. 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 [14] and practitioners [69] 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 [49; 55], security [13], reliability [19], and usability [42].

5.6 Analysing Requirements Models

A primary benefit of modeling requirements is the opportunity this provides for analysing them. Analysis techniques that have been investigated in RE include requirements animation [32], automated reasoning (e.g., analogical and case-based reasoning [54] and knowledge-based critiquing [23]), consistency checking (e.g., model checking [37]), and a variety of techniques for validation and verification (V&V) that we discuss in Section 7.

VI. 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 requirements 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 [18; 81]. From logic [3] to natural language [2], different languages have been shown to have different expressive and reasoning capabilities.

What is increasingly recognised as crucial, however, is *requirements management* – the ability, not only to write requirements but also to do so in a form that is readable and traceable by many, in order to manage their evolution over time. One attempt to achieve readability has been the development of a variety of documentation standards that provide guidelines for structuring requirements documents [78]. However, some authors, such as Kovitz [44], argue that standards or templates cannot in themselves provide a general structuring mechanism for requirements. Rather, he argues that the structure has to be developed for the particular context or problem in hand. Nevertheless, it is often the case that projects with rigid contractual constraints demand conformance to standards. Kovitz suggests some heuristics for focusing on the small details of writing requirements documentation, which can improve the quality of the requirements documentation, regardless of the format in which requirements are expressed.

Requirements traceability (RT) is another major factor that determines how easy it is to read, navigate, query and change requirements documentation. Gotel [31] defines requirements traceability as "the ability to describe and follow the life of a requirement in both forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)". RT lies at the heart of requirements management practice in that it can provide a rationale for requirements and is the basis for tools that analyse the consequences and impact of change. Providing RT in requirements documentation is a means of achieving integrity and completeness of that documentation, and has an important role to play in managing change, which will be discussed in Section 8.

VII. 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 [35] 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. Popper's observations on the limitations of empirical observation apply here : that scientific theories can never be proved correct through observation, they can only be refuted [61]. For RE, this view suggests that validation should adopt the same stance that software testers take: it should devise experiments to attempt to refute the current statement of requirements. Jackson [39] argues that descriptions used in RE should be refutable – those that are not refutable are vague, and should only be treated as *rough sketches*.

Logical positivism was severely criticized in the latter part of the twentieth century [5]. For example, Kuhn [45] observed that science tends to move through paradigm shifts, where the dominant paradigm determines the nature of current scientific theories. This leads to the realization that observation is not value-free, rather it is theory-driven, and is biased by the current paradigm. For requirements engineers, the methods and tools they use dominate the way that they see and describe problems. In the extreme case, this shifts the problem of validating requirements statements to a problem of convincing stakeholders that the chosen representation for requirements models is appropriate. Jackson captures this perspective through his identification of *problem frames* [39]. If stakeholders do not agree with the choice of problem frame, it is unlikely that they will ever agree with any statement of the requirements.

Ethno methodologists attempt to avoid the problem altogether, by refusing to impose modeling constructs on the stakeholders [30]. By discarding traditional problem analysis tools, they seek to apply value-free observations of stakeholder activities, and therefore circumvent the requirements validation issue altogether.

The second essential difficulty in requirements validation centers on the problem of disagreement among stakeholders. Recent approaches that explicitly model stakeholders' goal hierarchies make the problem clear: stakeholders may have goals that conflict with one another [79]. Requirements negotiation attempts to resolve conflicts between stakeholders without necessarily weakening satisfaction of each stakeholder's goals. Early approaches to requirements negotiation focused on modeling each stakeholder's contribution separately rather than trying to fit their contributions into a single consistent model [20] and on the importance of establishing common ground [70]. Boehm introduced the win-win approach [7] in which the win conditions for each stakeholder are identified, and the software process is managed and measured to ensure that all the win conditions are satisfied, through negotiation among the stakeholders.

The theory underlying these negotiation models is the same in each case: identify the most important goals of each participant, and ensure these goals are met. This approach is used in other RE techniques to promote agreement, without necessarily making the goals explicit [43]. For example, in Quality Function Deployment (QFD) [34], matrices are constructed to compare functional requirements with one another and rate their importance, but without explicitly identifying stakeholder goals.

We have described some essential difficulties in agreeing and validating requirements. These difficulties are compounded by a number of contextual issues, including contractual and procurement issues, and the fact that the political and social milieu in which the introduction of a new computer system changes the nature of work and the organizations [46].

VIII. 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 [9].

Changes to requirements documentation need to be managed. Minimally, this involves providing techniques and tools for configuration management and version control [22], and exploiting traceability 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* [6]. In any case, managing inconsistency [28] 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 [38].

While traceability 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 [21].

Managing changing requirements is not only a process of managing documentation, it is also a process of recognising change through continued requirements elicitation, re-evaluation 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 [4]. 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 [27].

IX. 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* [10] 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. Jacskon, for example, uses problem frames to structure different kinds of elementary and composite problems [39]. His argument is that identifying well-understood problems offers the possibility of selecting corresponding, appropriate, well-understood, solutions.

An alternative approach to organising, selecting and tailoring multiple methods is through the use of multiple perspectives or views of requirements [16; 26]. 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 [59].

Finally, to enable effective management of an integrated RE process, automated tool support is essential. Requirements management tools, such as DOORS [65], Requisite Pro [66], Cradle [77], and others, provide capabilities for documenting requirements, managing their change, and integrating them in different ways depending on project needs.

X. A REQUIREMENTS ENGINEERING ROADMAP

This paper has set out a roadmap, and we feel that no roadmap is complete without a big arrow labelled "you are here"[1]. By way of providing such a marker, we will summarise 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 [48]. 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:

- 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 contextualized enquiry techniques, including ethno methodology and participant observation [29; 63].
- 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* [84][2]. 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. This notion has been accompanied by a shift in emphasis away from modeling information flow and system state, and towards modeling stakeholders' goals [15] and scenarios that illustrate how goals are (or can be) achieved [51].
- The idea that the attempt to build consistent and complete requirements models is futile, and that RE has to take seriously the need to analyse and resolve conflicting requirements, to support stakeholder negotiation, and to reason with models that contain inconsistencies [28].

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:

1. Development of new techniques for formally modeling and analysing properties of the environment, as opposed to the 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.
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.
3. Richer models for capturing and analysing non-functional requirements.
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 analyse what impact a

particular architectural choice has on the ability to satisfy current and future requirements, and variations in requirements across a product family [27].

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 [50], and will facilitate the selection of commercial off-the-shelf (COTS) software [25; 53].
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 analyse 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.

XI. CONCLUSION

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.

REFERENCES

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

[2] Ambriola, V. & Gervasi, V. (1997). Processing Natural Language Requirements. *12th International Conference on Automated Software Engineering*, Lake Tahoe, USA, 3-5 November 1997, pp. 36-45.

[3] Antoniou, G. (1998). The role of nonmonotonic representations in requirements engineering. *International Journal of Software Engineering and Knowledge Engineering*, 8(3): 385-399.

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

[5] Blum, B. I. (1996). *Beyond Programming: To a New Era of Design*. Oxford: Oxford University Press.

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

[7] Boehm, B., Bose, P., Horowitz, E. & Lee, M. J. (1995). Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. *17th International Conference on Software Engineering (ICSE-17)*, Seattle, USA, 23-30 April 1995, pp. 243- 254.

[8] Boehm, B. W. (1981). *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

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

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

[11] Burg, J. F. M. (1997). *Linguistic Instruments in Requirements Engineering*. Amsterdam: IOS Press.

[12] Carter, R., Martin, J., Mayblin, B. & Munday, M. (1984). *Systems, Management and Change: A Graphic Guide*. London: Paul Chapman Publishing/Harper and Row.

[13] 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.

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

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

[16] Darke, P. & Shanks, G. (1996). Stakeholder Viewpoints in Requirements Definition: A Framework for Understanding Viewpoint Development Approaches. *Requirements Engineering*, 1(2): 88-105.

[17] Davis, A. (1992). Operational Prototyping: A New Development Approach. *Software*, 9(5): 70-78.

[18] Davis, A. (1993). *Software Requirements: Objects, Functions and States*. Prentice Hall.

[19] 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.

[20] Easterbrook, S. M. (1991). Resolving Conflicts Between Domain Descriptions with Computer-Supported Negotiation. *Knowledge Acquisition: An International Journal*, 3: 255-289.

[21] Easterbrook, S. M. & Nuseibeh, B. A. (1995). Managing Inconsistencies in an Evolving Specification. *Second IEEE Symposium on Requirements Engineering*, York, UK, March 27-29, pp. 48-55.

[22] Estublier, J. (2000). Software Configuration Management: A Roadmap.

[23] Fickas, S. & Nagarajan, P. (1988). Critiquing Software Specifications: a knowledge based approach. *IEEE Software*, 5(6).

[24] Finkelstein, A. (1993). Requirements Engineering: an overview. *2nd Asia-Pacific Software Engineering Conference (APSEC'93)*, Tokyo, Japan, 1993.

[25] Finkelstein, A., Ryan, M. & Spanoudakis, G. (1996). Software Package Requirements and Procurement. *8th International Workshop on Software Specification and design (IWSSD-9)*, Schloss Velen, Germany, pp. 141-146.

[26] Finkelstein, A. & Sommerville, I. (1996). The Viewpoints FAQ: Editorial - Viewpoints in Requirements Engineering. *Software Engineering Journal*, 11(1): 2-4.

[27] Garlan, D. (2000). Software Architecture: A Roadmap.

[28] Ghezzi, C. & Nuseibeh, B. (1998). Guest Editorial - Managing Inconsistency in Software Development. *Transactions on Software Engineering*, 24(11): 906-907.

[29] Goguen, J. & Jirotka, M. (Ed.). (1994). *Requirements Engineering: Social and Technical Issues*. London: Academic Press.
[30] Goguen, J. & Linde, C. (1993). Techniques for Requirements Elicitation. *1st IEEE International Symposium on Requirements Engineering (RE'93)*, San Diego, USA, 4-6th January 1993, pp. 152- 164.
[31] Gotel, O. & Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem. *1st International Conference on Requirements Engineering (ICRE'94)*, Colorado Springs, April 1994, pp. 94-101.
[32] Gravell, A. & Henderson, P. (1996). Executing Formal Specifications Need Not Be Harmful. *IEE Software Engineering Journal*, 11(2): 104-110.
[33] Greenspan, S. & Feblowitz, M. (1993). Requirements Engineering Using the SOS Paradigm. *1st International Symposium on Requirements Engineering (RE'93)*, San Diego, USA, 4-6 January 1993, pp. 260-263.
[34] Hauser, J. R. & Clausing, D. (1988). The House of Quality. *The Harvard Business Review* (3): 63-73.
[35] 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.
[36] Holtzblatt, K. & Beyer, H. R. (1995). Requirements Gathering: The Human Factor. *Communications of the ACM*, 38(5): 31-32.
[37] Holzmann, G. J. (1997). The Model Checker Spin. *Transactions on Software Engineering*, 23(5): 279-295.
[38] Hunter, A. & Nuseibeh, B. (1998). Managing Inconsistent Specifications: Reasoning, Analysis and Action. *ACM Transactions on Software Engineering and Methodology*, 7(4): 335-367.
[39] Jackson, M. (1995). *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. Addison Wesley.
[40] Jackson, M. & Zave, P. (1993). Domain Descriptions. *1st International Symposium on Requirements Engineering (RE'93)*, San Diego, USA, 4-6 January 1993, pp. 56-64.
[41] Jarke, M. & Kurki-Suonio, R. (1998). Guest Editorial - Special issue on Scenario Management. *IEEE Transactions on Software Engineering*, 24(12).
[42] Johnson, P. (1992). *Human-Computer Interaction: psychology, task analysis and software engineering*. TMH.
[43] Karlsson, J. & Ryan, K. (1997). Prioritizing Requirements Using a Cost-Value Approach. *IEEE Software*: 67-74.
[44] Kovitz, B. L. (1999). *Practical Software Requirements: A Manual of Contents & Style*. Manning.
[45] Kuhn, T. S. (1962). *The Structure of Scientific Revolutions*. Urbana: University of Chicago Press.
[46] Lehman, M. M. (1980). Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 1060-1076.
[47] Loucopoulos, P. & Kavakli, E. (1995). Enterprise Modeling and the Teleological Approach to Requirements Engineering. *International Journal of Intelligent and Cooperative Information Systems*, 4(1): 45-79.
[48] Loucopoulos, P. & Potts, C. (Ed.). (1996). *Requirements Engineering Journal*. Springer Verlag.
[49] Lutz, R., Helmer, G., Moseman, M., Statezni, D. & Tockey, S. (1998). Safety Analysis of Requirements for a Product Family. *3rd IEEE International Conference on Requirements Engineering (ICRE '98)*, USA, 6-10 April 1998, pp. 24-31.
[50] Maibaum, T. S. E. (2000). Mathematical Foundations of Software Engineering: A Roadmap.
[51] Maiden, N. (1998). CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, 5(4): 419-446.
[52] Maiden, N. & Rugg, G. (1996). ACRE: Selecting Methods For Requirements Acquisition. *Software Engineering Journal*, 11(3): 183-192.
[53] Maiden, N. A. M. & Ncube, C. (1998). Acquiring Requirements for Commercial Off-The-Shelf Package Selection. *IEEE Software*, 15(2): 46-56.
[54] Maiden, N. A. M. & Sutcliffe, A. G. (1992). Exploiting Reusable Specifications Through Analogy. *Communications of the ACM*, 34(5): 55-64.
[55] Modugno, F., Leveson, N. G., Reese, J. D., Partridge, K. & Sandys, S. D. (1997). Integrating Safety Analysis of Requirements Specifications. *3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Annapolis, USA, 6-10 January 1997, pp. 148- 159.
[56] Nakajo, T. & Kume, H. (1991). A Case History Analysis of Software Error Cause-Effect Relationships. *Transactions on Software Engineering*, 17(8): 830-838.
[57] Norman, D. A. (1993). Cognition in the Head and in the World: An Introduction to the Special Issue on Situated Action. *Cognitive Science*, 17(1): 1-6.
[58] Nuseibeh, B. (1997). Ariane 5: Who Dunnit? *IEEE Software*, 14(3): 15-16.
[59] Nuseibeh, B., Kramer, J. & Finkelstein, A. C. W. (1994). A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. *IEEE Transactions on Software Engineering*, 20(10): 760-773.
[60] Parnas, D. (2000). When to formalise. *Personal Communication* (Email), 17 February 2000.
[61] Popper, K. R. (1963). *Conjectures and Refutations: The Growth of Scientific Knowledge*. New York: Basic Books.
[62] Posner, M. I. (Ed.). (1993). *Foundations of Cognitive Science*. MIT Press.
[63] Potts, C. (1997). Requirements Models in Context. *3rd International Symposium on Requirements Engineering (RE'97)*, Annapolis, USA, 6-10 January 1997, pp. 102-104.
[64] Potts, C., Takahashi, K. & Anton, A. (1993). Inquiry-based requirements Analysis. *IEEE Software*, 11(2): 21-32.
[65] Quality Systems and Software (1999). DOORS <http://www.qss.co.uk/>
[66] Rational Corporation (1999). Requisite Pro <http://www.rational.com>
[67] Reubenstein, H. B. & Waters, R. C. (1991). The Requirements Apprentice: Automated Assistance for Requirements Acquisition. *IEEE Transactions on Software Engineering*, 17(3): 226-240.
[68] Robertson, S. & Robertson, J. (1994). *The Complete Systems Analysis: The Workbook, Textbook, Answers*. Dorset House.
[69] Robertson, S. & Robertson, J. (1999). *Mastering the Requirements Process*. Addison-Wesley.
[70] Robinson, W. N. & Volkov, S. (1998). Supporting the Negotiation Life-Cycle. *Communications of the ACM*, 41(5): 95-102.
[71] Saaltink, M. (1997). The Z/EVES System. *19th International Conference on the Z Formal Method (ZUM)*, Reading, UK, April 1997, LNCS 1212, pp. 72-88.
[72] Schneider, G. & Winters, J. (1998). *Applying Use Cases: a practical guide*. Addison-Wesley.
[73] Sharp, H., Finkelstein, A. & Galal, G. (1999). Stakeholder Identification in the Requirements Engineering Process. *Workshop on Requirements Engineering Processes*, Florence, Italy, 1-3 September 1999, pp. 387-391.
[74] Shaw, M. (1990). Prospects for an Engineering Discipline of Software. *IEEE Software*, 7(6): 15-24.
[75] Shaw, M. & Gaines, B. (1996). Requirements Acquisition. *Software Engineering Journal*, 11(3): 149-165.
[76] Stevens, R., Brook, P., Jackson, K. & Arnold, S. (1998). *Systems Engineering: Coping with Complexity*. Prentice Hall.

[77] Structured Software Systems Ltd (1999) CRADLE <http://www.threesl.com/>
[78] Thayer, R. & Dorfman, M. (Ed.). (1997). *Software Requirements Engineering* (2nd Edition). IEEE Computer Society Press.
[79] van Lamsweerde, A., Darimont, R. & Letier, E. (1998). Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 908-926.
[80] Viller, S. & Sommerville, I. (1999). Social Analysis in the Requirements Engineering Process: from ethnography to method. *4th International Symposium on Requirements Engineering (RE'99)*, Limerick, Ireland, 7-11th June 1999.
[81] Wieringa, R. J. (1996). *Requirements Engineering: Frameworks for Understanding*. Wiley.
[82] Yu, E. (1997). Towards Modeling and Reasoning Support for Early- Phase Requirements Engineering. *3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Annapolis, USA, 6-10 January 1997, pp. 226-235.
[83] Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, 29(4): 315-321.
[84] Zave, P. & Jackson, M. (1997). Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1): 1-30.

Dr. K. Sunil Manohar Reddy – Dr. K. Sunil Manohar Reddy is currently working as an Associate Professor in the Department of Computer Science and Engineering at Matrusri Engineering College, Hyderabad, India. His areas of interest are Neural Networks, Artificial Intelligence, Software Engineering, Cloud Computing, Computer Graphics and Data Warehousing and Data Mining. He has published and presented about 30 papers in National and International Conferences and Journals.