Requirements Engineering: a roadmap

Bashar Nuseibeh
Department of Computing
Imperial College
180 Queens' Gate
London SW7 2BZ, U.K.
ban@doc.ic.ac.uk

Steve Easterbrook
Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Ontario M5S 3H5, Canada
Email: sme@cs.toronto.edu

ABSTRACT
This paper presents an overview of the field of software systems requirements engineering (RE). It describes the main areas of RE practice, and highlights some related research issues.

1 Introduction

The primary measure of success of a software system is the degree to which it satisfies its customers. Customer satisfaction is determined by how closely the system meets a variety of stakeholder needs. Broadly speaking, *software systems requirements engineering* (RE) is the process of identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication and subsequent implementation. Such a process can present a number of difficulties. 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. Often, their goals and technical needs may not be explicit or may be difficult to articulate. And inevitably, their requirements may be constrained by a variety of factors outside their control.

This paper presents an overview of the main activities that constitute requirements engineering. While these activities are described independently and in a particular order, practical reality dictates that these activities are actually interleaved, iterative, and may span the entire software systems development life cycle. The paper is *not* a survey, it is a roadmap. Therefore, the related work cited in each section is only a guide to the reader interested in exploring a topic further. Nevertheless, the paper aims to provide sufficient content and direction to navigate the field of RE. It is organized as follows. Section 2 outlines the disciplines that provide the foundations for effective RE, while section 3 briefly describes the context and background needed in order to begin the RE process. Sections 4 to 8 describe the core RE activities:

- *eliciting* requirements,
- *modelling* and *analysing* requirements,
- *communicating* requirements,
- *agreeing* requirements, and
- *evolving* requirements.

Section 9 then discusses how these different activities may be integrated together in a single development process. The paper ends with some concluding remarks about research directions in the field.

2 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 on which it draws. Zave [71] 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 the "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 the stakeholders want, *defining* what designers have to build, and *verifying* that they have done so correctly upon delivery. Finally, the definition refers to specification "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. However, 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 cost-effectiveness can indeed be analysed. It also refers to the fact 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* [65], whose ultimate goal to is deliver some systems behaviour to its stakeholders. The special consideration that *software 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. If the focus of a project is on the development of software-intensive systems only, then one can indeed regard RE as purely the task of producing (software) descriptions [34].

Whether viewed at the systems level or the software level, RE is a multi-disciplinary, human-centred process. The tools and techniques used in RE draw upon a wide variety 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. Computer Science provides the framework to assess the feasibility of requirements and provides the building blocks of the descriptions to be produced. Although software engineering still lacks a mature science of software behaviour on which to draw, it is Computer Science that is at the forefront of developing such a science. Requirements engineers must draw on such a science in order to model the behaviour of the software they are specifying.

Since software is a formal description, analysis of its behaviour is amenable to formal reasoning. *Logic* provides a vehicle for achieving [2]. In RE, logic can be used to improve the rigour of the analysis performed, and to make the reasoning steps explicit. Different logics may be used to express different aspects of a required system. For example, temporal logic can be used to describe timing information, deontic logic to describe permissions and obligations, and linear logic to describe resources and their use. 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 very relevant to RE [65]. This includes work on characterizing systems, identifying their boundaries and managing their development life cycle [11, 69]. RE also encompasses work on systems analysis, traditionally found in the information systems world [59].

The context in which requirements engineering takes place is nearly always a human activity system, and the problem owners are people. 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 modelling requirements:

- *Cognitive psychology* helps to understand the difficulties people may have in describing their needs [17]. For example, problem domain experts often have large amounts of tacit knowledge, which is not amenable to introspection; hence their answers to questions posed by requirements analysts may not match their behaviour. Also, the requirements engineer may need to model the user's understanding of software user interfaces.
- *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. [25]. For example, the techniques of ethnomethodology [26] 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 organisation, and may affect the structure and communication paths within that organisation. A requirements gathering exercise can therefore become politicised. 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 [32].
- *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 improve understandability. Tools from linguistics can also be used in requirements elicitation, for instance to analyse communication patterns within an organisation [10].

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 modelling technique, because the choice of technique affects the set of phenomena that can be modelled, and may restrict what the requirements engineer can observe.

3 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 often also the case that requirements change during development and evolve after the 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 RE does occur early in the lifetime of projects, motivated by the evidence that requirements errors (e.g. misunderstood or omitted requirements) are more expensive to fix later in project lifecycles [18, 7, 48].

However, before a project can be started, some preparation in needed. Finkelstein [20] categorises such preparation as "Context" and "Groundwork". In particular, some assessment of the 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, and it is important that conflicts between high-level goals of an envisioned system are surfaced early, in order to establish a system's concept of operation and boundaries. Of course, risk assessment should be re-evaluated regularly throughout the development lifetime of a system [49], however, an initial evaluation is particularly critical.

4 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. Information gathered during requirements elicitation often has to be interpreted, analysed, modelled 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 modelling scheme (and vice versa: many modelling schemes imply the use of particular kinds of elicitation techniques).

Requirements to elicit. Most requirements engineers agree that one of the most important elicitation tasks is to establish a system's *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 most subsequent elicitation efforts. The elicitation of stakeholders and user classes, of goals and tasks, and of scenarios and use cases all follow from then on.

Identifying *stakeholders* – individuals or organisations who stand to gain or loose 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). Ideally, users should play a central role in the elicitation process, since they will be interacting with the delivered system. Often, however, other stakeholders (such as paying clients) are more influential, which can result in a significantly different system being delivered, to the one preferred by users. Of course, users themselves are not homogeneous, and part of the elicitation process is to identify different user classes (e.g., novice, expert, disabled, frequent, etc.) [63].

Goals denote the objectives a system must meet. Eliciting high level goals early on in the development process is crucial. However, goal-oriented requirements elicitation [13] 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 [16]. These tasks can often be represented in *use cases* that can be used to describe the outwardly visible requirements of systems [62]. 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 [36].

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. Often, the first port of call for a requirements elicitation exercise is *existing documentation*. This might include draft requirements documents, organisational charts, process models or standards, and user or other manuals of existing systems.

A number of other more structured techniques are also available to the requirements engineer. *Interviews*, for example, often provide an opportunity for an in-depth

exploration of issues relevant to one or a small number of stakeholders. *Questionnaires and surve*ys, on the other hand, can be used to reach a larger sample of stakeholders, while *meetings* allow focused brainstorming and early identification of conflicting requirements.

The use of *ethnography* to observe users in their 'normal' work environment is also becoming increasingly popular for elicitation of some requirements. It appears to be well suited when users find it difficult to articulate their needs and when requirements engineers are looking for a better understanding of the context in which a future system may be installed [68]. *Prototypes* on the other hand provide stakeholders with a concrete (although partial) model or system that they might expect to be delivered at the end of a development project [14]. Prototypes can be used to elicit stakeholder feedback in terms of missing requirements, unnecessary requirements, or confirmation (validation) that user requirements have been elicited satisfactorily. In fact, prototypes provide valuable feedback at many stages of the systems development process: during (pre-requirements) feasibility analysis, requirements elicitation, requirements validation, and exploratory design, to name a few.

Finally, techniques originally developed for knowledge acquisition for knowledge base systems can also be used for requirements elicitation [64]. Such techniques include *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), and *RAD/JAD workshops* (using consensus building workshops with unbiased facilitator) [44].

The elicitation process. With a large 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 [54] and CREWS [43] 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 [44].

5 Modelling and Analysing Requirements

Modelling – 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 (e.g., [15, 69]) focus almost entirely on modelling methods and their associated analysis techniques. Models can be used to represent a whole range of products of the RE process. Moreover, many modelling approaches are used as elicitation tools – where the modelling notation and partial models produced are used as drivers to prompt further information gathering.

The key question to ask for any modelling 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 modelling 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.

Enterprise modelling. The context of most requirements engineering activities and software systems is an *organisation* in which development takes place or in which a system will operate. Enterprise modelling and analysis deals with understanding an organisation'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 modelling is often used to capture the "purpose" of a system, by describing the behaviour of the organisation in which that system will operate [41]. This behaviour can be expressed in terms organisational objectives (or goals) and associated tasks and resources [70]. Others prefer to model an enterprise in terms of its business rules, work flows and the services that it will provide [29].

Modelling goals is particularly useful in RE. High-level business goals may be repeatedly refined as part of the elicitation process, leading to requirements that can then be operationalised [13].

Task analysis – an elicitation technique derived from work in human-computer interaction – is the process of discovering the way people perform their jobs. By understanding how people work, the requirements engineer can begin to identify areas of work that are problematic and that might need the support of an (automated) system [37].

The RE process uses and generates large volumes of information. This data needs to be understood, manipulated and managed. Data modelling provides the opportunity to represent information in the RE process, so that it may subsequently be analysed. Entity-Relationship-Attribute (ERA) type modelling is often used to represent data, however, increasingly, object-oriented models, such as Class Diagrams, are also being used.

Behavioural Modelling. Modelling requirements often involves modelling the dynamic or functional behaviour of stakeholders and systems (existing and required). A wide range of modelling methods are available for this purpose, from structured and object-oriented methods to Soft and or formal methods. These methods provide different levels of precision and are amenable to different kinds of analysis. Formal methods (for example, based on Z) can be difficult to construct, but are also amenable to automated analysis [61]. On the other hand, Soft methods provide "rich" representations [53] that non-technical stakeholders find appealing, but are often difficult to check automatically.

Domain Modelling. A significant proportion of the RE process is about developing "domain descriptions" [35]. A model of the domain provides an abstract description of the world in which an envisioned system will operate. Without such a description, it is difficult for the requirements engineer to understand the context of requirements and to identify opportunities for requirements reuse. Domain-specific models have also been shown to be essential for building automated tools, because they provide the ability to restrict analysis and reasoning, thereby making it tractable (e.g., [57]).

Modelling Non-Functional Requirements (NFRs). Quality, or non-functional, requirements are normally global attributes of a required system. They are generally regarded as more difficult requirements to express in a measurable way, making them more difficult to analyse. However, recent work by researchers [47] and practitioners [58] has emphasised the need and demonstrated the ability to model NFRs and express them in a form that is measurable or testable. There is also a growing body of research concerned with particular kinds of NFRs, such as safety [46, 42], security [12], reliability [23] and usability [37].

Analysing Requirements Models. A primary benefit of modelling requirements is the opportunity this provides for analysing them. Analysing software models is a wide ranging topic, however, for RE, a few areas are especially relevant. These include requirements animation [28], automated reasoning (e.g., analogical and case-based reasoning [45] and critiquing [19]) consistency checking (e.g., using model checking [33]), and a variety of techniques for validation and verification (V&V), which we discuss in section 7.

6 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, analysed, (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 [15, 69]. From Logic [4] to Natural Language [3], different languages have been shown to have different expressive and reasoning capabilities.

What is increasingly recognized as crucial, however, is *requirements management* – the ability, not only to write requirements but, to do so in a form that is readable and traceable by many. One attempt to achieve readability has been the development of a variety of documentation standards that provide guidelines for structuring requirements documents [67]. However, some authors (such as Kovitz [38]) 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 also suggests a variety of heuristics focusing on the small details of writing requirements documentation can improve the quality of the requirements documentation – regardless of the format in which requirements are expressed.

Requirements traceability (RT) is the another major factor that determines how easy requirements documentation is to read, navigate, query and change. Gotel [27] 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 most 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.

7 Agreeing Requirements

Having captured a statement of the requirements, it is vital that the all stakeholders agree with it. Recall that validation is the process of establishing that the requirements (model) elicited provides an accurate account of stakeholder requirements. Describing the requirements is an important step towards getting agreement. As Popper points out, "There is a world of difference between holding a belief or expecting something, and using human language to *say* so. The difference is that only if spoken out, and thus objectivized, does a belief become criticizable" [52]. Describing the requirements is a necessary precondition

not only for validating requirements, but 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 [31] 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 we covered all the aspects of the problem that the stakeholders regard as important.

Requirements validation is difficult for two important 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 modelled by building a consistent body of scientific 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. Karl Popper was one of the first philosophers to point out some the limitations of empirical observation. Popper's view was that scientific theories can never be proved correct through observation, but can only be refuted [51]. Proposing a new theory is therefore tantamount to inviting others to devise experiments to refute it. 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 [34] 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 criticised in the latter part of the twentieth century. Each attack offers illuminating insights for requirements engineers [5]. For example, Kuhn [39] 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. The process of empirical observation is biased by the current paradigm. For requirements engineers, this means that 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. Once again, Jackson captures this perspective through his identification of "problem frames" [34]. If the stakeholders do not agree with the choice of problem frame, it is unlikely that they will ever agree with any statement of the requirements. Ethnomethodologists attempt to avoid the problem altogether, by refusing to impose modelling constructs on the stakeholders [26]. 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 centres on the problem of disagreement between stakeholders. Recent approaches that explicitly model stakeholders' goal hierarchies make the problem clear: stakeholders may have goals that conflict with one another. In the KAOS approach for example, these are modelled as obstacles: the modelling process includes an explicit analysis of potential obstacles to each goal [40].

Requirements negotiation attempts to resolve conflicts between stakeholders without necessarily weakening satisfaction of each stakeholder's goals. Early approaches to requirements negotiation focused on the importance of establishing common ground [60], and on modelling each stakeholder's contribution separately rather than trying to fit their contributions into a single consistent model [17]. Boehm introduced the win-win approach [6] 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.

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. For example, in Quality Function Deployment (QFD) [30] matrices are constructed to compare functional requirements with one another and rate their importance, but without explicitly identifying stakeholder goals.

In summary, two essential difficulties in agreeing (and validating) requirements have been described: the first is philosophical and the second social. 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 organisations.

8 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 the RE [8].

As a first and minimal step, changing requirements documentation needs to be managed. This often involves providing techniques and tools for configuration management and version control, and exploiting traceability links to monitor and control the impact of changes in different parts of the documentation. Of course, 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 the systems in their operational environment.

For software systems, the primary kind of change that needs to be managed is change in software descriptions (specifications and other kinds of documentation). Specifications are generally changed for two reasons. Either the specification contains a problem that needs to be fixed (typically an inconsistency of some kind), or new requirements need to be added (as part of evolutionary development or to cope with changing stakeholder needs). Therefore, managing changing requirements is also a process of *managing inconsistency* [22] – whether this inconsistency is introduced by change or is as a result of change.

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) robust to change, and (b) flexible enough to be customized and adapted to changing requirements, is one of the key researches in software development.

9 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* [9] 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 modelling 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 [34]. His argument is that identifying well-understood problem, offers the possibility of selecting corresponding, appropriate, well-understood, solutions.

Another popular approach to RE is to support explicitly multiple perspectives or views of requirements [21]. This enables a wider range of requirements to be elicited, assists in identifying inconsistencies and conflicts between requirements, and can facilitate requirements partitioning and subsequent modelling and analysis. One particular "viewpoint-oriented" RE approach, explicitly supports both the development of multiple perspectives and the design and integration of multiple methods to support such an RE process [50].

Finally, to enable effective management of an integrated RE process, automated tool support is essential. Requirements management tools (such as DOORS [55], Requisite Pro [56], Craddle [1] and others) provide capabilities for documenting requirements, managing their change, and integrating them in different ways depending on project needs.

10 Concluding Remarks

Requirements engineering continues to be a crucial activity in any systems engineering process. The novelty of many software applications, the speed by which they need to be developed, and the degree to which they are expected to change, all play a role in determining how the RE process should be conducted. The demand for better, faster, and prettier software systems will continue, and the RE field will therefore continue to evolve to deal with different development scenarios.

Unfortunately, many delivered systems do not meet their customers' requirements, and this is, at least partly, due to ineffective RE (often treated as a time-consuming, contractual process).

Nevertheless, we believe that effective RE will continue to determine the success or failure of projects, and to determine the quality of systems that are delivered. However, we also believe that RE research should be focused increasingly on understanding and supporting more interleaved models of software development. In particular, the intertwining of requirements and design needs to be revisited [66], to provide software systems engineers with better guidance as they navigate the "process road" [24] between software requirements and software architectures.

Acknowledgements. This work was partially funded by the UK EPSRC projects MISE (GR/L 55964) and VOICI (GR/M 38582).

11 References

[1] 3SL, "CRADDLE", Structured Software Systems Ltd. http://www.threesl.com/, 1999.

[2] S. Abramsky, D. Gabbay, and T. Maibaum, Eds., *Handbook of Logic in Computer Science Vol 1: Background: Mathematical Structures*, Clarendon Press, 1992.

[3] V. Ambriola and V. Gervasi, "Processing Natural Language Requirements", Proc. of 12th Int. Conference on *Automated Software Engineering*, 36-45, Lake Tahoe, USA, IEEE CS Press, 3-5 Nov. 1997.

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

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

[6] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee, "Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", *Proc. of 17th Int. Conference on Software Engineering (ICSE-17)*, 243-254, Seattle, USA, IEEE CS Press, 23-30 April 1995.

[7] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[8] S. A. Bohner and R. S. Arnold, Eds., *Software Change Impact Analysis*, IEEE CS Press, 1996.

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

[10] J. F. M. Burg, *Linguistic Instruments in Requirements Engineering*, IOS Press, 1997.

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

[12] L. Chung, "Dealing with Security Requirements During the Development of Information Systems", *Proc. of 5th Int. Conference on Advanced Information Systems Engineering (CAiSE'93)*, 234-251, Paris, France, Springer-Verlag, 1993.

[13] A. Dardenne, A. v. Lamsweerde, and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, 203-50, Elsevier, 1993.

[14] A. Davis, "Operational Prototyping: A New Development Approach", *Software*, 9(5):70-78, IEEE CS Press, 1992.

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

[16] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*, 2nd Edition, Prentice Hall, 1998.

[17] S. M. Easterbrook, Ed., *CSCW: Cooperation or Conflict?*, Springer-Verlag, 1993.

[18] A. Endres, "An Analysis of Errors and Their Causes in System Programs", *Transactions on Software Engineering*, 1(2):140-149, IEEE CS Press, 1975.

[19] S. Fickas and P. Nagarajan, "Critiquing Software Specifications: a knowledge based approach", *Software*, 5(6):, IEEE CS Press, November 1988.

[20] A. Finkelstein, "Requirements Engineering: an overview", *Proc. of 2nd Asia-Pacific Software Engineering Conference (APSEC'93)*, Tokyo, Japan, IEEE CS Press, 1993.

[21] A. Finkelstein and I. Sommerville, "The Viewpoints FAQ: Editorial - Viewpoints in Requirements Engineering", *Software Engineering Journal*, 11(1):2-4, IEE/BCS, January 1996.

[22] C. Ghezzi and B. Nuseibeh, "Guest Editorial - Managing Inconsistency in Software Development", *Transactions on Software Engineering*, 24(11):906-907, IEEE CS Press, November 1998.

[23] D. D. Gobbo, M. Napolitano, J. Callahan, and B. Cukic, "Experience in Developing System Requirements Specification for a Sensor Failure Detection and Identification Scheme", *Proc. of 3rd High-Assurance Systems Engineering Symposium*, Washington, DC, IEEE CS Press, 13-14 Nov. 1998.

[24] M. Goedicke and B. Nuseibeh, "The Process Road Between Requirements and Design", *Proc. of 2nd World Conference on Integrated Design and process Technology (IDPT'96)*, 176-177, Austin, Texas, USA, SDPS, 1-4 December 1996.

[25] J. Goguen and M. Jirotka, Eds., *Requirements Engineering: Social and Technical Issues*, Academic Press, 1994.

[26] J. Goguen and C. Linde, "Techniques for Requirements Elicitation", *Proc. of 1st IEEE Int. Symposium on Requirements Engineering (RE'93)*, 152-164, San Diego, IEEE CS Press, 4-6th Jan. 1993.

[27] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem", *Proc. of 1st Int. Conference on Requirements Engineering (ICRE'94)*, 94-101, Colorado Springs, IEEE CS Press, April 1994.

[28] A. Gravell and P. Henderson, "Executing Formal Specifications Need Not Be Harmful", *Software Engineering Journal*, 11(2):104-110, IEE/BCS, March 1996.

[29] S. Greenspan and M. Feblowitz, "Requirements Engineering Using the SOS Paradigm", *Proc. of 1st Int. Symposium on Requirements Engineering (RE'93)*, 260-263, San Diego, IEEE CS Press, 4-6 Jan. 1993.

[30] J. R. Hauser and D. Clausing, "The House of Quality", *The Harvard Business Review*(3):63-73, May-June 1998.

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

[32] K. Holtzblatt and H. R. Beyer, "Requirements Gathering: The Human Factor", *Communications of the ACM*, 38(5):31-32, ACM Press, May 1995.

[33] G. J. Holzmann, "The Model Checker Spin", *Transactions on Software Engineering*, 23(5):279-295, IEEE CS Press, May 1997.

[34] M. Jackson, *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, Addison Wesley, 1995.

[35] M. Jackson and P. Zave, "Domain Descriptions", *Proc. of 1st Int. Symposium on Requirements Engineering (RE'93)*, 56-64, San Diego, USA, IEEE CS Press, 4-6 January 1993.

[36] M. Jarke and R. Kurki-Suonio, "Guest Editorial - Special issue on Scenario Management", *Transactions on Software Engineering*, 24(12):, IEEE CS Press, December 1998.

[37] P. Johnson, *Human-Computer Interaction: psychology, task analysis and software engineering*, McGraw-Hill, 1992.

[38] B. L. Kovitz, *Practical Software Requirements: A Manual of Contents & Style*, Manning, 1999.

[39] T. S. Kuhn, *The Structure of Scientific Revolutions*, University of Chicago Press, 1962.

[40] A. v. Lamsweerde and E. Letier, "Integrating Obstacles in Goal-Driven Requirements Engineering", *Proc. of 20th Int. Conference on Software Engineering (ICSE-20)*, 53-62, Kyoto, Japan, IEEE CS Press, 19-25 April1998.

[41] P. Loucopoulos and E. Kavakli, "Enterprise Modelling and the Teleological Approach to Requirements Engineering", *Int. Journal of Intelligent and Cooperative Information Systems*, 4(1):45-79, 1995.

[42] R. Lutz, G. Helmer, M. Moseman, D. Statezni, and S. Tockey, "Safety Analysis of Requirements for a Product Family", *Proc. of 3rd IEEE Int. Conference on Requirements Engineering (ICRE '98)*, 24-31, Colorado Springs, IEEE CS Press, 6-10 April 1998.

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

[44] N. Maiden and G. Rugg, "ACRE: Selecting Methods For Requirements Acquisition", *Software Engineering Journal*, 11(3):183-192, IEE/BCS.

[45] N. A. M. Maiden and A. G. Sutcliffe, "Exploiting Reusable Specifications Through Analogy", *Communications of the ACM*, 34(5):55-64, ACM Press, April 1992.

[46] F. Modugno, N. G. Leveson, J. D. Reese, K. Partridge, and S. D. Sandys, "Integrating Safety Analysis of Requirements Specifications", *Proc. of 3rd IEEE Int. Symposium on Requirements Engineering (RE'97)*, 148-159, Annapolis, IEEE CS Press, 6-10 Jan. 1997.

[47] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-functional Requirements: a process-oriented approach", *Transactions on Software Engineering*, 18(6):483-497, IEEE CS Press, June 1992.

[48] T. Nakajo and H. Kume, "A Case History Analysis of Software Error Cause-Effect Relationships", *Transactions on Software Engineering*, 17(8):830-838, IEEE CS Press, 1991.

[49] B. Nuseibeh, "Ariane 5: Who Dunnit?", *Software*, 14(3):15-16, IEEE CS Press, May 1997.

[50] B. Nuseibeh, J. Kramer, and A. C. W. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *Transactions on Software Engineering*, 20(10):760-773, IEEE CS Press, October 1994.

[51] K. R. Popper, *Conjectures and Refutations: The Growth of Scientific Knowledge*, Basic Books, 1963.

[52] K. R. Popper, "Campbell on the Evolutionary Theory of Knowledge", In *Evolutionary Epistemology, Rationality and the Sociology of Knowledge*, 115-120, G. Radnitsky and W. W. Bartley, Eds., La Salle, IL: Open Court, 1987.

[53] C. Potts, "Requirements Models in Context", *Proc. of 3rd Int. Symposium on Requirements Engineering (RE'97)*, 102-104, Annapolis, IEEE CS Press, 6-10 Jan. 1997.

[54] C. Potts, K. Takahashi, and A. Anton, "Inquiry-based requirements Analysis", *Software*, 11(2):21-32., IEEE CS Press, 1993.

[55] QSS, "DOORS", Quality Systems and Software http://www.qss.co.uk/, 1999.

[56] Rational, "Requisite Pro", Rational Corporation http://www.rational.com, 1999.

[57] H. B. Reubenstein and R. C. Waters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", *Transactions on Software Engineering*, 17(3):226-240, IEEE CS Press, March 1991.

[58] S. Roberston and J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, 1999.

[59] S. Robertson and J. Robertson, *The Complete Systems Analysis: The Workbook, The Textbook, the Answers*, Dorset House, 1994.

[60] W. N. Robinson and S. Volkov, "Supporting the Negotiation Life-Cycle", *Communications of the ACM*, 41(5):95-102, ACM Press, May 1998.

[61] M. Saaltink, "The Z/EVES System", *Proc. of 19th Int. Conference on the Z Formal Method (ZUM)*, 72-88, Reading, UK, Springer-Verlag, April 1997.

[62] G. Schneider and J. Winters, "Applying Use Cases: a practical guide," : Addison-Wesley, 1998.

[63] H. Sharp, A. Finkelstein, and G. Galal, "Stakeholder Identification in the Requirements Engineering Process", *Proc. of Workshop on Requirements Engineering Processes (REP'99) - DEXA'99*, 387-391, Florence, Italy, IEEE CS Press, 1-3 September 1999.

[64] M. Shaw and B. Gaines, "Requirements Acquisition", *Software Engineering Journal*, 11(3):149-165, IEE/BCS, May 1996.

[65] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *Systems Engineering: Coping with Complexity*, Prentice Hall Europe, 1998.

[66] W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation", *Communications of the ACM*, 25(7):438-440, ACM Press, July 1982.

[67] R. Thayer and M. Dorfman, Eds., *Software Requirements Engineering*,, 2nd Edition, IEEE CS Press, 1997.

[68] S. Viller and I. Sommerville, "Social Analysis in the Requirements Engineering process: from ethnography to method", *Proc. of 4th Int. Symposium on requirements Engineering (RE'99)*, Limerick, Ireland, IEEE CS Press, 7-11th June 1999.

[69] R. J. Wieringa, *Requirements Engineering: Frameworks for Understanding*, Wiley, 1996.

[70] E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering", *Proc. of 3rd IEEE Int. Symposium on Requirements Engineering (RE'97)*, 226-235, Annapolis, IEEE CS Press, 6-10 Jan. 1997.

[71] P. Zave, "Classification of Research Efforts in Requirements Engineering", *Computing Surveys*, 29(4):315-321, ACM Press, December 1997.