

# CHAPTER 1

## Introduction

---

### Introduction

Information systems are increasingly becoming an integral part of our everyday lives to the extent that the welfare of individuals, competitiveness of business concerns and effectiveness of public institutions often depend upon the correct and efficient functioning of these systems. The success of these systems, often referred to as socio-technical systems<sup>1</sup>, depends to a large extent in their ability to meet the needs and expectations of their *customer* (i.e. the individual, group or organisation that commissions the development of the system) as well as their *user* (i.e. the individual, group or organisation that will work with the system itself). It is therefore, the task of the *supplier* of the system (i.e. the system developer, or service provider) to deliver a solution that meets the expected level of functionality and ensures successful ‘integration’ of the technical system in the organisational setting.

It has long been established that the effectiveness and flexibility of a system are inexorably related to the correct understanding of the needs of the system’s customers or users. There is a

---

<sup>1</sup> i.e. systems that involve computer-based components interacting with people and other technical system components in an organisational setting.

key component therefore, of any development process which plays a central role in this process namely the *requirements specification*. The process of developing a requirements specification has been called *Requirements Engineering* [COMPUTER 1985; TSE 1977].

As a discipline Requirements Engineering is still evolving with a diversity of approaches being proposed and a lively debate going on. Therefore, it is neither possible nor appropriate to be prescriptive about *the* approach that one might adopt in developing a requirements specification. It is however important to discuss some of the concerns underpinning the field of Requirements Engineering and highlight some of the major issues of current investigation and practice before proceeding with a discussion on approaches to eliciting, representing and validating requirements.

To this end, this chapter first examines the term ‘requirements’ from two relevant perspectives, the organisational perspective and the software perspective. Section 1.1 briefly discusses the issue of requirements from an organisational perspective and in particular from the need of organisations to transform their functioning by using an information system as a facilitator to such a transformation. Furthermore, understanding the organisational setting is crucial to developing a more complete understanding of requirements for information systems. Information systems and their formal descriptions exist for some reason - they serve some strategic, tactical and operational objectives of the enterprise. Indeed the development of an information system impacts on the functioning and social organisation of the enterprise itself and therefore it is important to establish at the outset the significance that requirements have in an organisation context.

Section 1.2 considers requirements from a software engineering perspective and highlights the place of requirements in the software development lifecycle. This is the traditional view of examining requirements, considering that the task of developing a requirements specification precedes other development activities such as design and implementation.

To complete the discussion on requirements, section 1.3 examines the characteristics of the Requirements Engineering process itself. The processes involved in Requirements Engineering, as observed in controlled experiments and by analysis of industrial practices are discussed in this section. This gives an insight to the nature of problems that are specific to the task of requirements specification and analysis.

## 1.1 Requirements

A definition of requirements in [IEEE-Std.'610' 1990] is given as:

- (1) A condition or capacity needed by a user to solve a problem or achieve an objective.
- (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- (3) A documented representation of a condition or capability as in (1) or (2).

Although, this definition has been given with software systems in mind, it is general enough to apply to non-software specific situations.

In general, requirements fall into two broad categories: *market-driven* and *customer-specific*. These two broad categories of requirements have different characteristics and are often treated differently within a development process. Some of the key differences reported in an extensive study of requirements engineering projects ([Lubars, Potts, et al 1993]) are shown in figure 1.1.

Market-Driven Projects	Customer-Specific Projects
<ul style="list-style-type: none"><li>• Requirements are sketchy and informal.</li><li>• Use of techniques from manufacturers rather than Software Engineering, e.g. QFD.</li><li>• Specification is in the form of a marketing presentation.</li><li>• Not readily identifiable 'customer'. Developers tend to have less experience in application domain.</li><li>• Projects rely on consultants for advice on desirable features.</li><li>• Less structured approaches adopted. Task force used in 'brainstorming' sessions.</li></ul>	<ul style="list-style-type: none"><li>• Requirements are voluminous and more 'formal'.</li><li>• Use techniques from Software Engineering.</li><li>• Specification may be in hundreds of pages of documentation.</li><li>• Make use of domain expertise. Developers have in-depth knowledge of domain (even sometimes surpassing that of the customers).</li><li>• Projects rely on in-house personnel.</li><li>• Structured approach following a particular approach.</li></ul>

**Figure 1.1: Comparisons Between Market-Driven and Customer-Specific Projects**

The primary concern of this book is with ‘customer-specific’ requirements i.e. requirements for systems that will need to operate within a well identifiable organisational context, although, it should be stressed that some of the issues discussed in the book are equally applicable to ‘market-driven’ requirements for example, specification of the functionality of the intended product or identifying the enterprise objectives that motivate the development of a ‘product’.

### **1.1.1 Requirements - An Organisational Perspective**

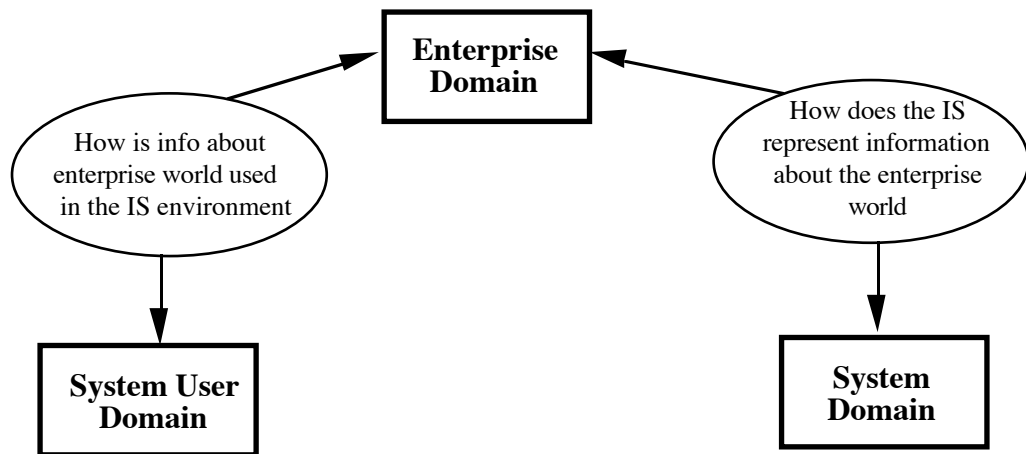
The usage of information systems has evolved from the automation of structured processes to applications that introduce change into fundamental business procedures. In increased level of sophistication, the contribution of information systems to organisations can be examined in terms of:

- *Automating* production by reducing the cost of the processes that make up production.
- *Informing* decision makers through the exploitation of automated processes.
- *Transforming* the organisation in a way that management and business processes make the best use of information technology by strategically aligning the information systems to the objectives of the organisation and its personnel.

In order for organisations to meet the challenges and opportunities presented by information systems in the automating, informing and in particular in the transforming stages the following are regarded as a general set of prerequisites [Morton 1991]:

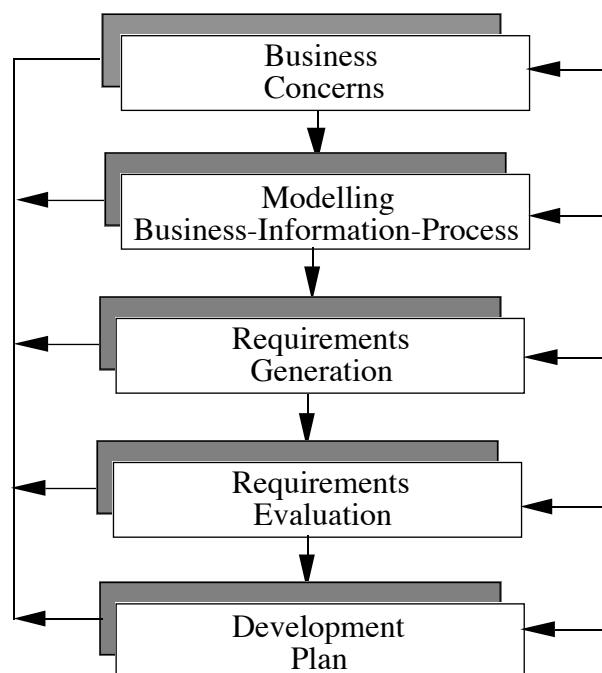
- Clear definition of business purpose and vision of what the organisation should become. This vision should be visible and understood by the organisation itself.
- Alignment of corporate strategy and information systems development.

In other words, the development of information systems is not simply about designing database structures and algorithms but also about understanding the needs of individuals and other stakeholders within the enterprise and ensuring that the system meets user requirements and business strategy. There is therefore, a natural relationship between the ‘enterprise’ domain and those of ‘system’ and ‘system user’ domains as shown in figure 1.2.



**Figure 1.2: Relating Information Systems to Organisations**

For example, by examining the objectives of the organisation, a rationale is established not only about the organisation itself but also about the infrastructure that supports or will support in the future the enterprise and the way that the system will fit the organisation and used by different end-user communities [Bubenko and Wangler 1993]. From an organisational perspective therefore, determination of requirements involves a number of interrelated tasks, shown in figure 1.3, that address management, social and information system concerns which need to be considered in a co-operative way in order to develop systems that fit their intended purpose.



**Figure 1.3: Requirements Specification from an Enterprise Perspective**

Information systems are entering a new phase moving beyond the traditional automation of routine organisational processes and towards the assisting of critical tactical and strategic enterprise processes. Development of such systems needs to concentrate on organisational aspects, delivering systems that are closer to the culture of organisations and wishes of individuals. Specifying requirements in this context directly address issues such as:

*Improving change management* by explicitly identifying and specifying those aspects of enterprises that are liable to change and by developing information systems that go beyond the automation of existing processes.

*Providing integration of views within an enterprise* by adopting an approach which encourages a co-operative generation and assessment of requirements.

*Relating information systems to business strategy* by facilitating the modelling of business goals and their realisation in information systems structure.

Many organisations perceive that a major challenge in the future will be to lead their organisations through the transformations necessary for a sustainable growth in a globally competitive environment. Competitiveness means increased quality, innovation and responsiveness to change. Business success is today critically dependent on the ability of the enterprises to link their information systems - their development and use - more closely to the business development process and such a success can only be achieved if infrastructure systems truly meet the needs and expectations as articulated within an organisational framework.

Crucially therefore, requirements engineering is about establishing the ‘connection’ between the need for some change within an organisational framework and the technology that could bring about such a change. In other words, requirements engineering can be considered as a way of managing change. This involves:

- an understanding at a conceptual level of the current status
- a definition of the change in terms of the transition from the ‘old’ conceptual situation to a ‘new’ target conceptual situation
- the implementation of the change in terms of the new components of the system and

- the integration of this new implementation in the environment which contained some legacy system.

### 1.1.2 Requirements - A Software Perspective

Interest about the role of requirements for software systems development can be traced to the early days of Software Engineering with the realisation that errors in the requirements definition stage resulted in costly maintenance of software systems at best and total rejection at worst [Bell and Thayer 1976]. As a consequence, Requirements Engineering was established as a sub field of Software Engineering with the task of developing models, techniques and tools that addressed this particular area. Since these early days the field has expanded in scope and point of view and there are many strands of investigation and practice nowadays that go beyond the strict confines of software construction.

Historically, the term Software Engineering was introduced when it became apparent that an engineering approach to software construction was needed. It was subsequently thought that if software development is to be approached in an engineering manner, then the same should apply to individual stages within it. As a result the term Requirements Engineering was adopted to describe an engineering approach to early stages of software construction.

The IEEE Glossary of Software-Engineering Terms gives the following definition of Software Engineering:

Software Engineering is a systematic approach to the development, operation, maintenance, and retirement of software

The above term implies the existence of a *life-cycle* view of software. According to this view, software generally undergoes the phases of development, operation, maintenance and retirement. Each of these phases can be seen from a dynamic viewpoint as a *process*. The term *process* which will be used throughout this book is defined according to the International Standards Organisation (ISO) as follows:

a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions

Software development, therefore, is a process which has as a purpose the development of a complete software system. Within software development there are events which correspond to the start of individual (sub) processes. Thus, requirements determination is a sub process within software development. Other processes within software development include design, coding and testing. Each of these processes has a unique purpose which, however, contributes to the overall purpose of developing the software. The purpose of design for example is, according to [IEEE-Std.'729' 1983]

...defining the software architecture (structure), components, modules, interfaces, test approach, and data for a software system to satisfy specified requirements

The view of software development as a sequence of processes is not dissimilar to other areas of engineering endeavour e.g. manufacturing. Manufacturing of a product involves a number of stages with processes such as product design, production planning, production monitoring and so on. Each manufacturing process has a purpose which contributes to the overall purpose i.e. the manufacturing of a complete product.

Manufacturing however is done in a systematic way. More specifically, there are three key elements-*methods*, *tools* and *procedures*, that enable the control of the manufacturing process and the development of a quality product.

- a *method* is a prescription of steps that need to be employed in order to achieve a specified result
- a *procedure* is a sequence of *actions* that must be performed
- a *tool* finally is the machinery used to perform some action as part of a procedure.

It is not difficult to draw analogies between software engineering and other engineering activities such as manufacturing. Similar to any other engineering discipline, software engineering uses methods, tools and procedures in a systematic way in order to arrive at the desired result which is the development of a complete software product.

Models for software development have been and are still continuing to be developed. With particular reference to the processes involved in requirements engineering, chapter 2 discusses a number of such process models.



There are many different models of software development. However, the majority of these different process models recognise the existence of components shown in figure 1.4 (note that no process model is shown in figure 1.4 but rather a set of deliverables - their derivation depends on the process model advocated by a particular method):

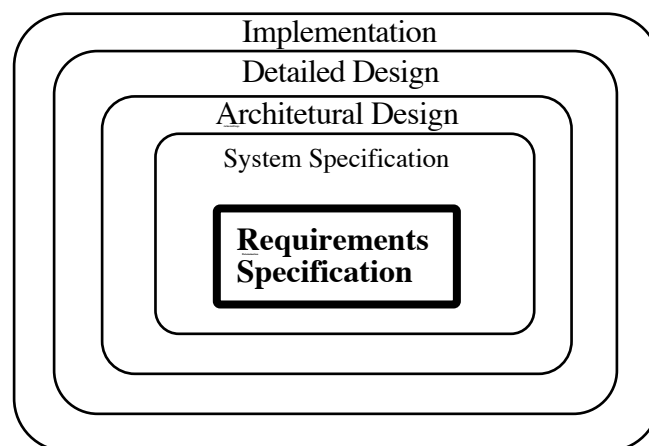
The *Requirements Specification* details the concerns of customers and users of the system, including the functionality of the system and the constraints that must be satisfied (both system and organisational constraints).

The *System Specification* is concerned with the definition of the system boundary and the information used in the interaction of the system and its environment. This specification represents a ‘black box’ view.

The *Architctural Design* represents a high-level view of the system’s internal design.

The *Detailed Design* represents a decomposition of the system and concentrates on the details of individual components.

The *Implementation* is concerned with the software components that finally realise the original user requirements.



**Figure 1.4: A General View of Components in Software Development**

The model shown in figure 1.4 recognises that a software development process consists of a number of distinct activities each activity yielding a particular category of ‘product’.

Developing a system constitutes a *design* activity. A design process typically involves [Dasgupta 1991]: (a) a set of requirements to be met by some artefact, (b) the output of the process is some design, (c) the goal of the designer is to produce a design such that if an implementation of this design were to materialise then the artefact would satisfy the requirements and (d) the designer has no knowledge of any design that satisfies the requirements. These properties of the design process are common to all information systems development approaches<sup>2</sup>.

The following characteristics are generally recognised in the software development process:

- the software development process involves the generation of a number of different models
- the software development process can be viewed as a series of steps
- these steps are goal driven and can be regarded as *transitions between representations*, preserving the semantic content, as refinements on these representations are applied.

The exact nature of the activities, the products and the transformations depends on the chosen process model.

Software is not different from other human artefacts in the sense that it is made in order to fulfil a perceived need or to solve a specific problem. If one considers some typical areas of today's world where software is being used, it becomes obvious that software is not an ultimate end *per se*, but rather a means to an end. Software is the *enabling* technology, i.e. the technology which helps people in their problem solving tasks, tasks which can be as straightforward as writing a letter on a word processor or as complicated as flying the Space Shuttle. Since software is being used in all sorts of conceivable applications by all kinds of people, more often than not, the need for a software solution is experienced by people who are not software expert themselves. Naturally, non software experts cannot build the software they want themselves, at least in the vast majority of cases and therefore their need for a software solution must be catered for by the experts, the software developers. It is this seemingly straightforward situation, but in fact laden with many problems and difficulties, where a software user requires

---

<sup>2</sup> Although the degree to which different methods adhere to these four aspects may vary considerably.

the services of a software provider which gave birth to terms like 'software crisis' and to the emergence of Requirements Engineering as a discipline.

At first, having a non-expert requiring the services of an expert seems like a totally natural situation which is often encountered in every day life. To those who have been involved with software either as users or as builders, the situation regarding the provision of software that satisfies some requirement, where this requirement is part of a larger set of organisational needs and expectations, would be recognised as sufficiently different to everyday needs for a number of reasons:

- Because software is not made of any physical substance it cannot be described in standard physical descriptions such as material, colour, dimensions etc.; it is rather described in terms of the often intangible characteristics of the situations, tasks and environments in which people use it.
- Software users best understand and describe their own work; software experts are more familiar with their domain-software.

It becomes apparent now that the seemingly straightforward task of understanding one's requirements and translating them into a software solution is a far cry from being as such, i.e. straightforward. The requirements part of software development is what is termed a *hard* problem, and yet one which we can ill-afford to leave unsolved. One obvious reason why requirements are important is because they set the criteria for the acceptance, success and usefulness of the software that is to be built. As discussed already, software on its own has no particular value; it is only when used as an enabler to other activities that it acquires a value. The best engineered piece of software is therefore worthless to someone if the software cannot be utilised to address their problem solving needs.

In addition, developing software is an activity which taxes a scarce resource, namely human software developers. Statistics show that while our ability to produce software will be increasing at a rate of 4% a year this is outstripped by the demand for new software which is increasing with an annual rate of 12%. In contrast the rate of increase in the number of qualified software developers is only 4% a year. It is obvious that we cannot afford the development of useless software and the waste of scarce resources such as time, people and money. A systematic approach to software requirements is needed, which will ensure the understanding of the user requirements and the production of useful software in a cost

effective way. Such an approach has to follow an *engineering* approach i.e. to apply proven methods, techniques and tools in a well described fashion.

## 1.2 Requirements Specification

Documenting requirements for software construction is an activity which results in what has been traditionally termed *requirements specification*. According to [Rzepka and Ohno 1985] a requirements specification represents both a model of what is needed and a statement of the problem under consideration. There is a wide variety of ways of expressing a requirements specification, ranging from informal natural language text to more formal graphical and mathematical notations. The structure of the specification itself varies according to different standards and practices [DOD-STD-2167A 1988; IEEE-Std.'830' 1984; NCC 1987].

At this point it is important to address the question “what is the purpose of a requirements specification?”. There are a number of reasons for striving to develop a requirements specification. First, it provides a focal point for the process of trying to correctly understand the needs of the customer and user of the intended system. In other words it is the target of a systematic approach with the specification itself relying on the use of some ‘language’ for representing the contents of the specification. Second, the specification can and should be the means by which a potentially large and diverse population of requirements stakeholders and requirements analysts communicate. The specification itself can be used for clarifying a situation about the intended system or its environment i.e. the organisational context. Third, the specification may be part of contractual arrangements, a situation that may become especially relevant when an organisation wishes to procure a system from some vendor rather than develop it ‘in house’. Fourth, the specification can be used for evaluating the final product and could play a leading role in any acceptance tests agreed between system consumer and supplier. Irrespective of its intended use the need for developing a requirements specification which at the very least expresses the problem in hand is well accepted by practitioners.

The traditional view of a requirements specification is that of a *functional* specification i.e. a definition of the desired service of the intended system. For example requirements for a system that handles a customer transaction in an airline ticket reservation system would need to address issues such as “what is the information required by the system in order to issue a ticket and what results will the transaction processing function

will yield?”. This view of specification is indeed prominent in many contemporary information systems methods<sup>3</sup>.

A functional requirements specification is concerned, as the name implies, with the description of the fundamental functions of the software components that make up the system. One is interested in defining the transformations which the system components should perform on inputs in order to produce some output. Functions are therefore, specified in terms of *inputs*, *processing* and *outputs*. A dynamic view of a system’s functionality would need to consider aspects such as *control* (e.g. sequencing and parallelism), *timing* of functions (e.g. starting and finishing), as well as the behaviour of the system in terms of *exceptional* situations. Inevitably, since functions deal with a variety of data formats, data will also need to be defined and form part of the functional specification. Data can correspond to inputs and outputs to functions, stored data and transient data. The specification of ‘real-world’ *entities* and their *relationships*, particularly for data-centred systems, need to be defined at appropriate levels of abstraction and related to descriptions of system functions.

The emergence of software development methods during the late 1970’s and 1980’s gave prominence to the importance of functional specifications. A variety of specification languages<sup>4</sup> have been developed and extensively used for industrial and commercial projects. However, this over-reliance on functional specifications has been criticised as having a number of undesirable side effects. One concern is the amount of detail with which both requirements holders and requirements analysts have to deal. It has been argued c.f. [McDermid 1994] that when a functional specification becomes the focal point of requirements analysis then one makes a decision on the boundary of the system before any understanding is gained of the real needs of the requirements holders. In other words, functional specifications tend to deflect attention from other important aspects. For example, it is at least as important as defining a functional specification to consider issues such as the objectives of the system itself and the relationship of these to organisational objectives or specifying other desirable properties of the system (e.g. performance, security, usability, etc.) and constraints on its development (e.g. use of a particular toolset, economic constraints etc.). It is only then that one can adequately understand the reasoning behind the needs and aspirations of requirements holders. Furthermore, such a wider view of a requirements specification could accommodate situations requiring resolution of conflict of requirements at an organisational level, deciding to give

---

<sup>3</sup> For a discussion on functional requirements models as found in contemporary methods the interested reader may refer to [COMPUTER 1985; Olle, Sol, et al 1983; Olle, Sol, et al 1984; Olle, Sol, et al 1986].

<sup>4</sup> There exist for example a number of formalisms that are ideally suited to this task.

priorities to the stated requirements, or evaluating alternative scenarios for the satisfying of the stated requirements.

Another area of concern is the relationship between a requirements specification and a system architecture. It has long been accepted that a requirements specification should define the ‘what’ i.e. a description of the problem in hand and not the ‘how’ i.e. the way that the problem is to be solved. It has been argued in the past that a requirement specification should just say enough about the problem and nothing else. In this sense the requirements engineering process is a front-end to a series of other activities within the domain of software development. There are a number of factors however, that make a distinction between the ‘what’ and the ‘how’ difficult to achieve and in many situations possibly even inappropriate. There is much anecdotal evidence that in practice there are many projects in which some understanding of the system architecture is required in order to be able to articulate, represent and evaluate requirements that by their very nature address the solution space, albeit at an architectural rather than a detailed design or software implementation level. For example, a requirement on some system characteristic (e.g. “the display screen of an air traffic control system should be capable of handling up to 100 tracks”) or a requirement on some general architectural consideration (e.g. “the system must conform with existing practice for client-server organisation” or “the system must conform to some communication standard”) are all important requirements that cannot be ignored until the detailed design stage since by their very nature impose constraints on the design itself. Furthermore, many intended systems have to be considered in the framework of other legacy systems and developers very frequently have little choice on infrastructure components.

The example statements above can be thought as architectural requirements i.e. requirements which inevitably are imposed on the solution by the customer or user of the system. These statements may be considered as qualifiers on some ‘real’ organisational problem (e.g. “the need for air traffic controllers to visualise air traffic scenarios”) but there would be little justification in this example to exclude this type of requirement from the specification. In the context of Requirements Engineering, the relationship between the ‘what’ and the ‘how’ is nowadays not as clear-cut as traditionally thought. A number of authors advocate that a requirements specification needs to go beyond the description of functionality and performance issues and that by including architectural issues during requirements provides an early opportunity to consider important parameters, tangible ones such as cost, as well as intangible ones such as acceptability of the system, in the introduction or evolution of a system [Garlan 1994; Jackson 1994; McDermid 1994; Mead 1994; Reubenstein 1994; Shekaran 1994].

The purpose of building a software system is to be found outside the system itself, in the *enterprise*<sup>5</sup>, i.e. the context in which the system will function. Requirements of customers need to be represented in a specification in terms of the explicitly stated (in the specification) observed phenomena about the enterprise itself [Bubenko, Rolland, et al 1994; Bubenko and Wangler 1993; Greenspan, Mylopoulos, et al 1994; Jackson 1994; Jackson and Zave 1993; Loucopoulos and Katsouli 1992; Loucopoulos, McBrien, et al 1991; Nellborn, Bubenko, et al 1992; Yu and Mylopoulos 1994; Yu 1993].

A broader view therefore, of requirements specification is one that goes beyond the description of system functionalities. A functional specification should be one view supplemented, or even motivated by two other perspectives. First, an understanding should be gained of the domain within which the intended system will be firmly embedded. An explicit definition of the enterprise within which the system will eventually operate is a fundamental prerequisite to the development of a common understanding of the requirements holders, system customers, system users and system developers about the problem in hand. The emerging consensus within the Requirements Engineering community is that a requirements specification should include not only software specifications but also any kind of information describing ‘real world’ phenomena. Second, an understanding should be gained of the constraints that can be placed on the system, its environment or its development, known as *nonfunctional* requirements (NFRs) (e.g. security, availability, portability, usability, performance, etc.). For example the requirement that the “airline booking system must respond within 15 seconds” would be a non-functional requirement that needs to be considered by the system designer who would have to make a number of design choices in order to meet this constraint<sup>6</sup>.

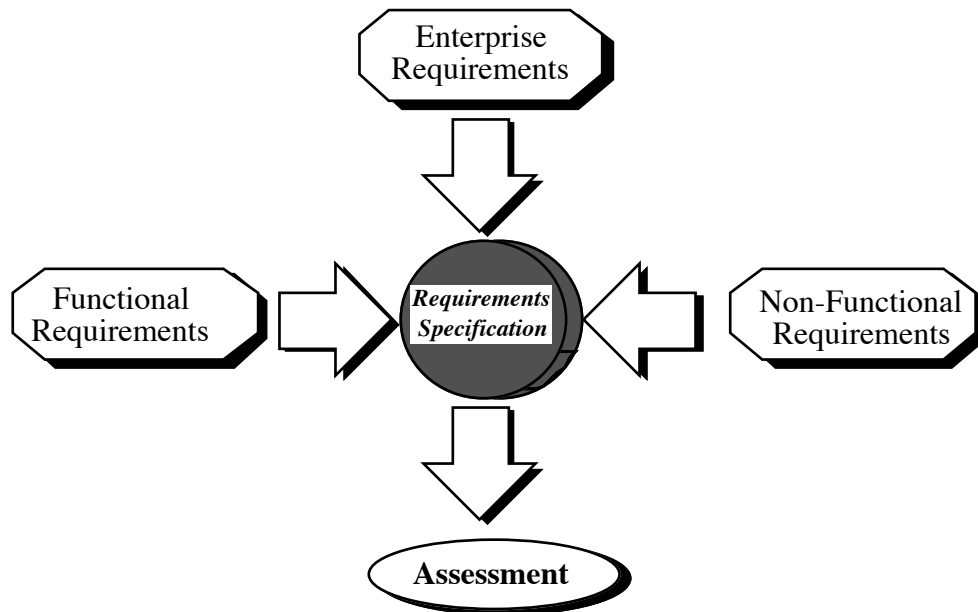
The term ‘requirements specification’ is used throughout this book from the broader perspective to refer to a description of requirements in the enterprise expressed in terms of the phenomena that are common to the enterprise and system domains. Descriptions of the enterprise are independent of any behaviour of any system whereas descriptions of the system refer to properties that the system must provide. This view is depicted in figure 1.5, where a

---

<sup>5</sup> Synonyms of this are the terms ‘application domain’ and ‘environment’.

<sup>6</sup> The distinction between functional and non functional requirements is not often clear and some authors prefer to avoid this distinction. It has been rightly argued, that some requirements after originally being classified as non functional they become, in due course, functional requirements themselves. For example, in an air traffic control system, the need to be able to handle some upper limit of aircraft tracks would be originally expressed as a NFR but this eventually will have to be handled by a system function. However, from a methodological perspective the distinction is useful in delienating different areas of concern during requirements analysis and therefore, NFRs are considered from a distinct viewpoint in this book.

requirements specification is shown to constitute an interrelated set of descriptions in three domains namely, the enterprise, functional and non-functional domains.



**Figure 1.5: Conceptual Framework for Requirements Specifications**

### 1.3 Requirements Engineering

The term *requirements engineering* (RE) can be defined as “the systematic process of developing requirements through an iterative co-operative process of analysing the problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained”.

This definition reflects the view that requirements specification involves an interplay of concerns between representation, social and cognitive aspects [Pohl 1993]. Issues of representation range from informal descriptions such as natural language expressions and hypertext to formal conceptual modelling languages. In the social domain, consideration is given to the complex social process in which the communication and co-operative interaction between the stakeholders of the requirements determines the quality of the final product. Issues in the cognitive domain concern different orientations of models in terms of understanding the process itself and validating the requirements.

Requirements Engineering consists of the knowledge elicitation, representation and validation cycle. The success of the requirements engineering process often depends on the ability *to*



*proceed from informal, fuzzy individual statements of requirements to a formal specification that is understood and agreed by all stakeholders.* However, the process is far from deterministic and straight forward. A requirements specification cannot be developed in a simple, linear fashion but a cyclic approach which gradually yields an involving specification seems to be more appropriate.

The transition from *informal to formal* requirements constitutes a conceptualisation activity within which a developer might make use of domain knowledge partly expressed in descriptions of the enterprise, and partly in existing requirements specifications. Reflecting back *from formal to informal* requirements is a process of validation which may take a number of different forms including prototyping, and explanation as to the decisions made in producing a requirements specification.

It is a truism that there is no unique or standard way for specifying requirements and although a number of authors have argued that specifications need to conform to certain characteristics such as completeness, correctness, unambiguity, understandability, modifiability and consistency [Dorfman and Thayer 1990], many of these qualities are hard to achieve. For example, it is very hard to test for completeness since there is no other ‘model’ against which the specification can be tested; it is only through repeated validation cycles that one can gain some confidence of completeness. The process is typically situation, context, and issue dependent, i.e. the kind of specification that is being developed during a development step depends on the development issues and questions experienced in previous steps.

Whilst, most of the skills required to develop software are primarily technical in nature this is not sufficient for the task of eliciting, specifying and validating requirements. The need to understand the underlying skills for the task of systems requirements analysis and their relationship to successful job performance has involved the use of other disciplines, apart from Computer Science, in attempting to define the way that systems analysts perform certain tasks. For example, the task of systems requirements analysis can be viewed as a type of *problem solving* [Newel and Simon 1972]. Under this perspective, requirements analysis is the reasoning process which attempts to understand the requirements of a problem domain in order to synthesise a solution for a system which will satisfy the needs of its users. During this thought process, an analyst may use clues, goals, strategies, heuristics, hypotheses, information and knowledge which has been acquired from different sources, i.e. the problem domain as well as from the analyst’s own experience.

Examination of various Requirements Engineering projects has provided insights into the problem itself. For example, the following observations have been made about requirements and requirements analysis [Vitalari and Dickson 1983] [Button and Sharrock 1994]:

- Analysis problems, at their inception, have ill-defined boundaries, structure, and a sufficient degree of uncertainty about the nature and make-up of the solution.
- Requirements are found in organisational contexts, with associated conflicts on expectations and demands about some intentional system.
- The solutions to analysis problems are artificial. That is, they are designed and hence many potential solutions exist for any one problem. Ending the process of requirements specification is a matter of practical necessity.
- Analysis problems are dynamic. That is, they change while they are being solved because of their organisational context and the multiple participants involved in the definition and specification process.
- Solutions to analysis problems require interdisciplinary knowledge and skill.
- The knowledge base of the systems analyst is continually evolving and the analyst must be ready to incorporate changes in the technology and to participate with users in different ways.
- The process of analysis itself, is primarily cognitive in nature, requiring the analyst to structure an abstract problem, process diverse information, and develop a logical and internally consistent set of specifications. All the other skills such as interpersonal interaction and organisational skill facilitate this cognitive process.

A number of empirical studies have also examined the way that the Requirements Engineering process is carried out within the software engineering domain. A study of software engineering practices, in many different application areas, reports that “accurate problem domain knowledge is critical to the success of a project” and “requirements volatility causes major difficulties during development” [Curtis, Krasner, et al 1988].

The uncertainty inherent in trying to ‘discover’ and document requirements is also problematic. A review of the state of practice in Requirements Engineering [Lubars, Potts, et al 1993] revealed that “although most informants were able to describe the nature of requirements specification that they produced, they were unable to describe the *process* by which they arrived at these specifications”. Also, “in customer-specific projects, changes to the requirements occurred due to changes in the environment whereas in market-driven projects competing products and insights to the market affected requirements. The importance of tracking the effects of changes to requirements through designs and implementation was recognised by most organisations”.

By observing the way that requirements analysts carry out their work and by comparing experienced analysts to novices the following characteristics emerged [Curtis, Krasner, et al 1988; Fickas 1987; Sutcliffe 1990; Sutcliffe and Maiden 1990; Sutcliffe and Maiden 1989; Vitalari and Dickson 1983].

- Analysts use information from the environment to classify problems and relate them to previous experience. Experienced analysts begin by establishing a set of context questions and then proceed by considering alternatives. Much of the contextual information depends on previous knowledge about the application domain and the analogies that an analyst will establish are based on such knowledge.
- Expert analysts tend to start solving a problem by forming a mental model of the problem at an abstract level. This model is then refined, by a progression of transformations, into a concrete model, as further information is obtained.
- Hypotheses are developed as to the nature of a solution, as information is collected. Experienced analysts use hypothetical examples to capture more facts from a requirements holder. Such examples are also used to clarify some previously acquired information about the object system.
- Developers almost always summarise in order to verify their findings. It has been observed that during a typical user-analyst session the analyst will summarise two or three times and each time the summarisation will trigger a new set of questions.

A requirements specification is likely to change many times before proceeding to design and as discussed already, a specification needs to be subjected to evaluation in order to gain

confidence as to its validity. This cyclic approach of acquisition-representation-evaluation involves a succession of propositions which are increasingly closer to end users' perceptions about the target system.

A requirement for an improved system begins with an initial hypothesis which is vague and requires further elaboration in order to produce the desired result. In this sense, requirements analysis involves the generation of hypotheses, and subjecting these hypotheses to a process of disconfirmation [Aguero and Dasgupta 1987; Hooton, Aguero, et al 1988]. These hypotheses are generated and evaluated against attributes of the intended system which it is anticipated that it will meet a desired state in the enterprise. Hypothesis formulation is based on the vision or belief that the participants in the analysis process may hold about the intended system, its role in the enterprise, its effects on organisation practices, its effects on personal and group status and so on. In short the participants define a set of system characteristics which in their opinion will lead to an improved situation. Hypotheses evaluation is based on the idea that every hypothesis undergoes criticism and is thoroughly examined for its validity i.e. looking for evidence to disconfirm the hypothesis. This implies that a specification describing objects, processes business rules, agents, IS components and so on, represents a set of propositions that are considered to be true until proven otherwise.

## **Summary**

Requirements Engineering is the activity that transforms the needs and wishes of customers and potential users of computerised systems, usually incomplete, and expressed in informal terms, into complete, precise, and consistent specifications, preferably written in formal notations.

This activity is arguably the most crucial activity in system development, if only because errors made in the early requirements specification phases are the most costly to repair once the system has been implemented. It is also a most delicate one, as it requires heavy involvement of requirements stakeholders, and a consensus between them, as well as between requirements stakeholders and system designers, who have quite different backgrounds and concerns.

Requirements engineering is a systematic process, normally carried out within a broader spectrum of development activities, that attempts to discover, capture and document the requirements of many different stakeholders, those that commission the project as well as those that will eventually use the product. The input to this process is usually an informal, ill-defined 'wish-list' whereas the output should be a specification which is formal enough to be analysed

and agreed upon by requirements stakeholders and developers alike. Such a specification should have three orientation:

- a view of key aspects of the enterprise defining the objectives for the target system
- a view of the required functionality of the system and
- a view of the properties that must be exhibited by the system and its environment for it to meet the enterprise objectives.

The above issues have been examined from a number of different disciplines and indeed different ‘philosophical’ standpoints.

Areas of concern, which ultimately influence the approach adopted in deriving a requirements specification, fall into two broad categories:

- issues of functionality and service to be provided by the intended system and
- issues of suitability of the intended system in an organisational context.

Traditionally, these two classes of concern have given rise to two broad classes of approaches, the so call ‘hard’ and ‘soft’ methods. This rather unhelpful distinction has tended to polarise the way that one considers requirements determination with the effect that important facets of requirements go missing from specifications. However, recent realisation that requirements for socio-technical systems span the areas of concern of both ‘hard’ and ‘soft’ methods has resulted in the investigation of techniques that go beyond the traditional divide c.f. [Jirotka and Goguen 1994; Petrie 1992]. Emerging techniques within the field of Requirements Engineering advocate a balanced view between technical and organisational considerations.

## References

- Agüero, U. and Dasgupta, S. (1987)** *A Plausibility-Driven Approach to Computer Architecture Design*, Communications of the ACM, Vol. 30, No. 11, 1987, pp. 922-931.
- Bell, T.E. and Thayer, T.A. (1976)** *Software Requirements: Are they Really a Problem*, 2nd International Conference on Software Engineering, 1976, pp. 61-68.
- Bubenko, J., Rolland, C., Loucopoulos, P. and de Antonellis, V. (1994)** *Facilitating "Fuzzy to Formal" Requirements Modelling*, IEEE International Conference on Requirements Engineering, 1994.
- Bubenko, J.A. and Wangler, B. (1993)** *Objectives Driven Capture of Business Rules and Information Systems Requirements*, IEEE Conference on Systems, Man and Cybernetics, 1993.
- Button, G. and Sharrock, W. (1994)** *Occasioned practices in the Work of Software Engineers*, in 'Requirements Engineering: Social and Technical Issues', M. Jiroka and J. A. Goguen (ed.), Academic Press, London, pp. 217-240.
- COMPUTER (1985)** *Special Issue on Requirements Engineering*, IEEE Computer, 1985.
- Curtis, B., Krasner, H. and Iscoe, N. (1988)** *A Field Study of the Software Design Process for Large Systems*, CACM, Vol. 31, No. 11, 1988, pp. 1268 ff.
- Dasgupta, S. (1991)** *Design Theory and Computer Science*, Cambridge University Press, Cambridge, UK, 1991.
- DOD-STD-2167A (1988)** *Defense System Software Development*, Department of Defence, February, 29, 1988, 1988.
- Dorfman, M. and Thayer, R.H. (1990) (ed.)** *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, California.

- Fickas, S. (1987)** *Automating the Analysis Process: An Example*, 4th International Workshop on Software Specification and Design, Moterey, USA, 1987.
- Garlan, D. (1994)** *The Role of Software Architecture in Requirements Engineering - Position Statement*, The 1st International Conference on Requirements Engineering, IEEE Computer Society Press, Colorado Springs, Colorado, 1994, pp. 240.
- Greenspan, S., Mylopoulos, J. and Borgida, A. (1994)** *On Formal Requirements Modeling Languages: RML Revisited*, 16th International Conference on Software Engineering (ICSE-94), IEEE Computer Science Press, 1994, pp. 135-148.
- Hooton, A., Aguero, U. and Dasgupta, S. (1988)** *An Exercise in Plausibility-Driven Design*, IEEE Computer, Vol. 27, No. 7, 1988, pp. 21-33.
- IEEE-Std.'610' (1990)** *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers, New York, Std.610.12-1990, 1990.
- IEEE-Std.'729' (1983)** *IEEE Standard 729. Glossary of Software Engineering Terminolgy*, The Institute of Electrical and Electronics Engineers, New York, 1983.
- IEEE-Std.'830' (1984)** *IEEE Guide to Software Requirements Specifications*, The Institute of Electrical and Electronics Engineers, New York, ANSI/IEEE Std 830-1984, 1984.
- Jackson, M. (1994)** *The Role of Software Architecture in Requirements Engineering - Position Statement*, The 1st International Conference on Requirements Engineering, IEEE Computer Society Press, Colorado Springs, Colorado, 1994, pp. 241.
- Jackson, M. and Zave, P. (1993)** *Domain Descriptions*, IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, California, 1993, pp. 56-64.
- Jirotko, M. and Goguen, J. (1994) (ed.)** *Requirements Engineering: Social and Technical Issues*, Computers and People Series, B. R. Gaines and A. Monk (ed.), Academic Press, London.

- Loucopoulos, P. and Katsouli, E. (1992)** *Modelling Business Rules in an Office Environment*, ACM SIGOIS, No. August, 1992.
- Loucopoulos, P., McBrien, P., Schumacker, F., Theodoulidis, B., Kopanas, V. and Wangler, B. (1991)** *Integrating Database Technology, Rule-Based Systems and Temporal Reasoning for Effective Information Systems: the TEMPORA Paradigm*, Journal of Information Systems, Vol. 1, No. 2, 1991, pp. 129-152.
- Lubars, M., Potts, C. and Richter, C. (1993)** *A Review of the State of the Practice in Requirements Modelling*, IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, California, 1993, pp. 2-14.
- McDermid, J.A. (1994)** *Requirements Analysis: Orthodoxy, Fundamentalism and Heresy*, in 'Requirements Engineering: Social and Technical Issues', M. Jirotko and J. A. Goguen (ed.), Academic Press, London, pp. 17-40.
- Mead, N.R. (1994)** *The Role of Software Architecture in Requirements Engineering - Position Statement*, The 1st International Conference on Requirements Engineering, IEEE Computer Society Press, Colorado Springs, Colorado, 1994, pp. 242.
- Morton, M.S. (1991)** *The Corporation of the 1990s: Information Technology and Organisational Transformation*, Oxford University Press, Oxford, 1991.
- NCC (1987)** *The STARTS Guide. A guide to methods and software tools for the construction of large real-time systems*, National Computing Centre Ltd, Manchester, UK, 1987.
- Nellborn, C., Bubenko, J. and Gustafsson, M. (1992)** *Enterprise Modelling - the Key to Capturing Requirements for Information Systems*, SISU, F3 Project Internal Report, 1992.
- Newel, R. and Simon, H. (1972)** *Human Problem Solving*, Prentice-Hall, 1972.
- Olle, T.W., Sol, H.G. and Tully, C.J. (1983) (ed.)** *Information Systems Design Methodologies : A Feature Analysis*, North Holland, Amsterdam.
- Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (1984) (ed.)** *Information System Design Methodologies: A Comparative Review*, North Holland, Amsterdam.



- Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (1986) (ed.)** *Information Systems Design Methodologies : Improving the Practice*, Elsevier Science Publishers B.V (North Holland), Amsterdam.
- Petrie, C.J. (1992) (ed.)** *Proceedings of the 1st Conference on 'Enterprise Integration Modeling'*, Scientific and Engineering Computation Series, MIT Press, Cambridge, Massachusetts & London, UK.
- Pohl, K. (1993)** *The Three Dimensions of Requirements Engineering*, 5th International Conference on Advanced Information Systems Engineering (CAiSE•93), C. Rolland, F. Bodart and C. Cauvet (ed.), Springer-Verlag, Paris, France, 1993, pp. 275-292.
- Reubenstein (1994)** *The Role of Software Architecture in Requirements Engineering - Position Statement*, The 1st International Conference on Requirements Engineering, IEEE Computer Society Press, Colorado Springs, Colorado, 1994, pp. 244.
- Rzepka, W. and Ohno, Y. (1985)** *Requirements Engineering Environments: Software Tools for Modelling User Needs*, IEEE Computer, No. April, 1985, 1985.
- Shekaran, M.C. (1994)** *The Role of Software Architecture in Requirements Engineering - Position Statement*, The 1st International Conference on Requirements Engineering, IEEE Computer Society Press, Colorado Springs, Colorado, 1994, pp. 245.
- Sutcliffe, A. (1990)** *Human Factors in Information Systems: a Research Agenda and some Experience*, Human Factors in Information Systems Analysis and Design, A. Finkelstein, M. J. Tauber and R. Traummuller (ed.), North-Holland, Scherding, Austria, 1990, pp. 5-23.
- Sutcliffe, A. and Maiden, N. (1990)** *Analysing the analyst: Requirements for the next generation of CASE tools*, in 'The Next generation of CASE-tools', S. Brinkemper and G. wijers (ed.), SERC, Noordwijkerhout, The Netherlands, pp. C2-1 - 9.
- Sutcliffe, A.G. and Maiden, N.A.M. (1989)** *Analysing the Analyst: Cognitive Models in Software Engineering*, , 1989.
- TSE (1977)** *Special Issue on Requirements Engineering*, IEEE Transactions on Software Engineering, 1977.

- Vitalari, N.P. and Dickson, G.W. (1983)** *Problem Solving for Effective Systems Analysis: An Experimental Exploration*, Communications of the ACM, Vol. 26, No. 11, 1983.
- Yu, E. and Mylopoulos, J. (1994)** *Understanding ‘Why’ in Software Process Modeling, Analysis and Design*, 16th International Conference on Software Engineering, Sorrento, Italy, 1994.
- Yu, E.S.K. (1993)** *Modelling Organizations for Information Systems Requirements Engineering*, IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, California, 1993, pp. 34-41.