CHAPTER 2

QUALITY REQUIREMENT SPECIFICATION

2.1 Introduction

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, *software systems requirements engineering* is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. There are a number of inherent difficulties in this process. Stakeholders (including paying customers, users and developers) may be numerous and distributed. Their goals may vary and conflict, depending on their perspectives of the environment in which they work and the tasks they wish to accomplish. Their goals may not be explicit or may be difficult to articulate, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control.
2.1.1 Definition of Requirement Engineering

Zave provides one of the clearest definitions:

“Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software. 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” [1].

This definition is attractive for a number of reasons. First, it highlights the importance of “real-world goals” that motivate the development of a software system. These represent the ‘why’ as well as the ‘what’ of a system. Second, it refers to “precise specifications”. These provide the basis for analysing requirements, validating that they are indeed what stakeholders want, defining what designers have to build, and verifying that they have done so correctly upon delivery. Finally, the definition refers to specifications’ “evolution over time and across software families”, emphasizing the reality of a changing world and the need to reuse partial specifications, as engineers often do in other branches of engineering.

In reality, software cannot function in isolation from the system in which it is embedded, and hence Requirement Engineering has to encompass a systems level view. We therefore prefer to characterize Requirement Engineering as a branch of systems engineering [2], whose ultimate goal is to deliver some systems behavior to its stakeholders. The special consideration that software systems requirements engineering has received is largely due to the abstract and invisible nature of software, and the vast range and variety of
problems that admit to software solutions. Application of systems theory and practice is also relevant to Requirement Engineering [2]. This includes work on characterizing systems, identifying their boundaries and managing their development life cycle [3,4]. Requirement Engineering also encompasses work on systems analysis, traditionally found in the information systems world [15].

2.1.2 Important Roles of Requirement Engineering

- Requirement Engineering is often regarded as a front-end activity in the software systems development process. This is generally true, although it is usually also the case that requirements change during development and evolve after a system has been in operation for some time. Therefore, Requirement Engineering plays an important role in the management of change in software development. Nevertheless, the bulk of the effort of Requirement Engineering does occur early in the lifetime of a project, motivated by the evidence that requirements errors, such as misunderstood or omitted requirements, are more expensive to fix later in project lifecycles [5,6]. Before a project can be started, some preparation is needed. Finkelstein categorizes such preparation as context and groundwork [7]. In the past, it was often the case that Requirement Engineering methods assumed that it was performed for a specific customer, who could sign off a requirements specification.
• However, Requirement Engineering is actually performed in a variety of contexts, including market-driven product development and development for a specific customer with the eventual intention of developing a broader market. The type of product will also affect the choice of method.

• Requirement Engineering for information systems is very different from Requirement Engineering for embedded control systems, which is different again from Requirement Engineering for generic services such as networking and operating systems. For groundwork, some assessment of a project’s feasibility and associated risks needs to be undertaken, and Requirement Engineering plays a crucial role in making such an assessment. It is often possible to estimate project costs, schedules and technical feasibility from precise specifications of requirements. It is also important that conflicts between high-level goals of an envisioned system surface early, in order to establish a system’s concept of operation and boundaries. Of course, risk should be re-evaluated regularly throughout the development lifetime of a system, since changes in the environment can change the associated development risks.

Groundwork also includes the identification of a suitable process for Requirement Engineering, and the selection of methods and techniques for the its various activities. We use the term process here to denote an instance of a process model, which is an abstract description of how to conduct a collection of activities, describing the behavior
of one or more agents and their management of resources. A *technique* prescribes how to perform one particular activity - and, if necessary, how to describe the product of that activity in a particular notation. A *method* provides a prescription for how to perform a collection of activities, focusing on how a related set of techniques can be integrated, and providing guidance on their use. Methods provide a systematic approach to combining different techniques and notations, and *method engineering* [14] plays an important role in designing the Requirement Engineering process to be deployed for a particular problem or domain. Methods provide heuristics and guidelines for the requirements engineer to deploy the appropriate notation or modeling technique at different stages of the process.

2.1.3 Functions

Requirement Engineering is a multi-disciplinary activity, deploying a variety of techniques and tools at different stages of development and for different kinds of application domains.

- Requirements engineering is concerned with the goals, desired properties and constraints of complex systems that involve software systems, organizations and people. It also covers how requirements relate to business processes, soft issues, work redesign, system and software architecture and testing. This process is regarded as one of the most important aspects of building an information system because it is during
this process that it is decided what is to be built [9]. In early studies, Bell and Thayer observed that inadequate, inconsistent, incomplete or ambiguous requirements are numerous and have a critical impact on the quality of resulting software [10]. They concluded that “the requirements for a system do not arise naturally; instead they need to be engineered and have continuing reviews and revisions”. Boehm estimated that the late correction of requirements errors could cost up to 200 times as much as correction during such requirement engineering. Brooks stated that “the most important function that a software builder performs for the client is the iterative extraction and refinement of the product requirements” [11].

• Requirement Engineering is the process of determining what is to be produced in a software system. In developing a complex software system, the requirements engineering process has the widely recognized goal of determining the needs for, and the intended external behavior, of a system design. This process is regarded as one of the most important parts of building a software system: "The hardest single part of building a software system is deciding what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No part of the work so cripples the resulting systems if done wrong. No other part is more difficult to rectify later"[12].
• Requirement Engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system [13].

• Requirement Engineering is also concerned with interpreting and understanding stakeholder’s terminology, concepts, viewpoints and goals. Hence, it must concern itself with an understanding of beliefs of stakeholders, the question of what is observable in the world, and question of what can be agreed on as objectively true. Such issues become important whenever one wishes to talk about validating requirements, selecting a modeling technique etc. It is a multi-disciplinary activity, deploying a variety of techniques and tools at different stages of development and for different kinds of application domains.

2.2 Requirement Engineering Phases

Software requirements express the needs and constraints that are placed upon a software product that contribute to the satisfaction of some real world application, alternately, the properties that must be exhibited in order to solve some real world problem. Requirement Engineering is a 30-year-old term, designed to describe the actual process of answering the
most important question of any software project. Developing software is traditionally divided into four phases: requirement analysis, design, code and testing. Having answered poorly during the first phase, that of the requirements, can result to unsatisfactory software and therefore cause huge extra costs, needed to make changes into already completed software. The purpose of Requirement Engineering is to make sure one does end up with the right and correctly formatted answers in the form of requirements. One of the most difficult problems in deciding on the system requirements is the communication gap between different stakeholders. The system analysts tend to speak with technical terms while the customer and end-users prefers to use the language used in their daily business. As a result the analysts may not have a clear idea what kind of system they are building and the customers and end-users are often surprised when they notice that the final product does not fulfill their expectations and solve the problems they needed the system for the first place.

Careful analysis and validation of requirements have been generally accepted as important ingredients for the success of software systems development. The inherent complexity of establishing an unambiguous, complete, and consistent set of requirements prior to system design and construction has led software engineering researchers and practitioners to propose and adopt techniques which can provide a framework to identify important and useful requirements. Since the success of a software system depends on how well the requirements are being engineered, Requirement Engineering remains an eminent part of engineering process. Requirement Engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the
specification and managing the requirements as they are transformed into an operational system. Requirement Engineering is also concerned with interpreting and understanding stakeholder’s terminology, concepts, viewpoints and goals. Hence, Requirement Engineering must concern itself with an understanding of beliefs of stakeholders, the question of what is observable in the world, and question of what can be agreed on as objectively true. Such issues become important whenever one wishes to talk about validating requirements, selecting a modeling technique etc. Requirement Engineering is often regarded as a front-end activity in the software system development process. It plays an important role in the management of change in software development.

Requirements Engineering is an important aspect of any software project, and is a general term used to encompass all the following phases related to requirements:

### 2.2.1 Requirement Abstraction and Elicitation

Software requirements arrive in different shapes and forms to development organizations. This is particularly the case in market-driven requirements engineering, where the requirements are on products rather than directed towards projects. This result in challenges related to making different requirements comparable. Abstraction allows the placement of requirements on different levels and supports the mechanism to break down the requirements to make them comparable to each other. The elicitation of requirements is perhaps considered as first important activity in Requirement Engineering process.
Requirement Elicitation involves sifting through large amounts of information and deciding what exactly is relevant. Information gathered from Requirement gathering phase often has to be interpreted, analyzed, modeled and validated before the requirement engineer can feel confident that a complete enough set of requirements of a system have been collected. One of the most important goals of elicitation is to find out what problem needs to be solved and hence identify system boundaries. Goals denote the objectives the system must have. Eliciting high level goals early in the development process is crucial.

Eliciting goals focuses the requirement engineer on the problem domain and needs of the stakeholders, rather than on possible solution to problems. It is often the case that users find it difficult to articulate the requirements to this end, a requirement engineer can resort to eliciting information about the task users currently perform and those that they might want to perform [20]. A number of elicitation techniques are available to the requirement engineer. Each method itself has its strengths and weakness and is normally best suited for use in particular application domain. Of course, in some circumstances a full blown method may be neither required nor necessary. Instead, the requirements engineer needs to select the appropriate technique or techniques most suitable for the elicitation in hand. In such situations, technique selection guidance is more appropriate than a rigid method [21].
2.2.2 Requirement Analysis

Requirement Analysis is a process of analyzing requirements to detect and resolve conflicts between requirements, discover the bounds of the system and how it must interact with its environment, elaborate system requirements to software requirements. Requirement analysis clearly represents problem requirements and leads smoothly to further phases in “needs” to be built. Requirement analysis caters for the required context, behavior and properties of a problem to be clearly identified [14]. Analysis results in the specification of software’s operational characteristics; indicates software’s interface with other system elements and establishes constraints that software must meet. Requirements analysis provides the software designer with a representation of information, function and behavior that can be translated to architectural, interface and component-level designs. In few software projects, certain things reoccur across all projects within a specific application domain. These are called as analysis patterns and represent something within the application domain that can be reused in many applications. These analysis patterns speed up the analysis process by suggesting design patterns and reliable solution to common problems [15].

System Requirements Analysis can be a challenging phase, because all of the major stakeholders and their interests are brought into the process of determining requirements. The quality of the final product is highly dependent on the effectiveness of the requirements identification process. Since the requirements form the basis for all future work on the project, from design and development to testing and documentation, it is of the utmost importance that the Project Team creates a complete and accurate
representation of all requirements that the system must accommodate. Accurately identified requirements result from effective communication and collaboration among all members of the Project Team, and provide the best chance of creating a system that fully satisfies the needs of the Customers.

The primary goal of this phase is to create a detailed Functional Specification defining the full set of system capabilities to be implemented, along with accompanying data and process models illustrating the information to be managed and the processes to be supported by the new system. The Functional Specification will evolve throughout this phase of the development process as detailed business requirements are captured, and as supporting process and data models are created, ensuring that the eventual solution provides the customers with the functionality they need to meet their stated business objectives.

### 2.2.3 Requirement Specification

A requirement specification is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies *at a particular pointing time* (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other’s requirements from that perspective at a given point in time.

The requirement specification document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web
site, and so on) must provide, as well as states any required constraints by which the system must abide. It also functions as a blueprint for completing a project with as little cost growth as possible. It is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it. A well-designed, well-written specification accomplishes four major goals:

- **It provides feedback to the customer.** A requirement specification is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, it should be written in natural language, in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.

- **It decomposes the problem into component parts.** The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.

- **It serves as an input to the design specification.** As mentioned previously, it serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, it must contain sufficient detail in the functional system requirements so that a design solution can be devised.
• **Requirement specification serves as a product validation check.** It also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

A specification can be a written document, a set of graphical models, a formal mathematical model or a collection of these. It is sometimes necessary to remain flexible when a specification is to be developed. A precise specification may be an informal document which is a work product produced by a requirement engineer. It will serve as a foundation for further activities like evaluation with stakeholders etc. An informal requirement specification will informally document the function, scope, requirements etc. of a computer based system and the constraints that will govern its development.

2.2.4 Requirement Validation

Requirements validation is a critical phase of requirements engineering processes, which makes sure that requirements are correct, consistent, complete and accurate. Requirements validation is used in determining the right requirements. The main objective of validation is to certify that requirement specification document is the acceptable description of the system, which is going to be implemented. The process of validating the requirements ensures three things: (a) “the set of requirements is correct, complete, and consistent”, (b) a model for satisfying the requirements can be created, and (c) “a real-world solution” for testing the requirements can be built to make sure that requirements are satisfying the stakeholder’s needs.

Requirements validation is very important process especially in case of critical systems where we cannot afford problems. Critical systems
are generally embedded systems, which are in-charge of supervision, signaling, or controlling tasks; these activities are performed by interacting with their environment through sensors and actuators [16]. For example: computer based control system for operation theater devices, or a computer based railway crossing are critical systems. Thousands of people’s lives may depend on these systems. Requirements for these systems should be properly validated according to their particular scenarios. The cost of finding and fixing requirements problems makes difference here. According to Boehm, finding and fixing cost of requirements problems is one hundred times more expensive after delivery of the product as compared to during requirements and early design phases [17].

According to Firesmith, incorrect or poorly-specified requirements create many problems during system integration, operational testing, manufacturing or deployment of the system [18]. Requirements are not always properly validated by their stakeholders; as a result end product is not acceptable for most of the stakeholders, even if it is verified by testing department. Fixing of requirements problems may have negative impacts on schedule and cost of the project, and some functionality may be missing upon release.

Organizations are facing problems in validating the requirements due to limited time and other considerations; validation of requirements is done in-formally either on ad-hoc bases or simple peer reviews [18, 19]. The selection of appropriate requirements validation techniques for validating the requirements is very important.

The work products produced in the precise specification are assessed for quality during this step. The stakeholders examine the specification to ensure that all the software requirements have been
stated unambiguously; those inconsistencies, omissions, errors have been detected and corrected and that the product conforms to the standards established for the process. This evaluation establishes that the requirements specified provide an accurate account of stakeholder requirements. Explicitly describing the requirements is a necessary precondition not only for validating requirements but also for resolving conflicts between stakeholders. An essential difficulty in requirements validation centers on the problem of disagreement among stakeholders. Requirement negotiation attempts to resolve conflicts between stakeholders without necessarily weakening satisfaction of stakeholder’s goal.

2.3 Proposed Framework for Quality Requirement Specification
Requirements development is an iterative process. One should not expect to go through the steps in a one-shot, linear fashion. For example, the requirements analysts may talk to a user and then analyze what the user had to say. They may go back to that user for clarification and then document what they understand as that part of the requirements. They may then go on to talk to another user, or hold a joint requirements workshop with several user representatives. Their analysis may then include building a prototype that they show to a focus group. Based on that information the analyst documents additional requirements and holds a requirements walk-through to validate that set of requirements. The analyst then moves on to eliciting the requirements for the next feature and so on. After one or more iterations through the software requirements development process, part or all of the requirements are deemed “good
enough” to baseline and become the basis for software design and development.

2.3.1 SRS Framework

Requirement Engineering Framework includes all the stages of Requirement Engineering arranged in an iterative manner to achieve an effective and efficient product.

Fig 2.1: SRS Framework

In the above defined framework, i.e Fig.2.1, all the phases of Requirement Engineering process are arranged in a sequence inhibiting an iterative behavior. This framework will produce those requirements which are actually necessary and intended by the
stakeholders. By applying an iterative behavior to these phases, a repeated execution can improve the quality of requirements by subsequent evaluation by stakeholders. This process first gathers requirements from various sources and collects them at one place for further engineering using the process of fusion. Many different approaches have been proposed for requirement gathering. For example, meetings are conducted attended by both software engineers and stakeholders; an agenda is suggested to cover all the important points, rules for preparation and participation are established etc.

### 2.2.2 Analysis of SRS Framework:

*Requirement fusion* is a process of collecting the needs to solve a problem or issues and achieve an objective. It is not quite accurate to say that requirements are in the minds of the clients; it would be more accurate to say that they are in a social system of client organization. They need to be invented and that invention has to be a cooperative venture involving the clients, the users and the developers. All the requirements gathered from various sources are fused together at one place. Important aspect of fusion is that it will collect the related and similar requirements and store them in groups.

This phase will enable the next phase i.e. *elicitation*, to select the intended requirements from the predefined groups of related requirements. Necessary and useful requirements are then elicited using various elicitation techniques. A requirement engineer can resort to eliciting information about the tasks, users currently
perform and those they might want to perform which can be often represented in use cases, used to describe outwardly visible requirements of a system. More specifically, the requirement engineer may choose a particular path through a scenario, in order to better understand some aspects of using a system. The identified requirements are then analyzed. Elicitation will help in identifying goals, determining scope and identifying functional and non-functional requirements.

**Analysis** deals with understanding an organization’s structure; the business rules that affect its operation; the goals, tasks and responsibilities of its constituent members; the data that it needs, generates and manipulates. Modeling is often used to capture the purpose of a system, by describing the behavior of the organization in which that system will operate. Modeling goals are particularly useful in Requirement Engineering. Elaboration is an analysis modeling action. Elaboration is driven by the creation and refinement of user scenarios that describe how the end user will interact with the system.

To sum up, analysis will help to model requirements and data, determine feasibility and to refine requirements.

An informal *specification* is created that specifies all the necessary requirements for the development process and the final product. It will document customer as well as developer’s requirements and give a specification of the interaction among all.

The created description/specification is evaluated by the stakeholders and users to identify any lack of requirements or any ambiguous requirements being included in the specification. The problems of inconsistency, completeness and unambiguity are
addresses so that these can be again analyzed and an appropriate set of requirements can be created. These changes defined by the stakeholders are communicated and the defined requirements are again reviewed and refined. Validation will also include the usability check and will provide some test cases to simulate requirements. This refinement of requirement will include the analysis phase where the requirements will again be analyzed according to the changes made by the stakeholders.

This iterative process goes on until the requirements gathered are satisfactory and in accordance to the specification of various stakeholders. If the requirements identified are satisfactory, then the final product of Requirement Engineering i.e. formal specification of requirement can be submitted to the system for use in further phases. This specification is the final product achieved after executing various phases of Requirement Engineering iteratively. This framework provides a product in the form of a formal document of requirements which will ensure the requirement veracity and will be an input to the next phase of the process life cycle. Since Requirement Engineering is an important phase in software development life cycle, this framework will be applicable to all process models of Software Engineering.

2.4 Eliciting Quality Requirements

Software requirements have been repeatedly recognized during the past years to be a real problem. Bell and Thayer observed that inadequate, incomplete, inconsistent and ambiguous requirements have a critical impact on the resulting software [22]. He concluded that “requirements for
a system do not arise naturally; instead, they need to be engineered and have continuing review and revision”. Boehm estimated that the late correction of requirements errors could cost up to 200 times as much as correction during requirement engineering [23].

Requirement Engineering is concerned with the identification of the goals to be achieved by the envisioned system, the operationalization of such goals into services and constraints, and the assignment of responsibilities for the resulting requirements to humans, devices and software. Getting high quality requirements is difficult and crucial. Recent surveys have confirmed the growing recognition of Requirement Engineering as an area of utmost importance in software engineering research and practice. A great deal of formal evidence exists that the typical quality of actual requirements specification today is very poor. This poor quality of individual requirements and requirement specification that document them is a primary reason why so many projects continue to fail. Unfortunately the poor quality of requirements is typically not recognized during requirement engineering and the evaluation of requirement specification.

2.4.1 Quality Requirement Parameters

The classical notation of requirements quality focuses on correctness, unambiguity, completeness, focus, minimal i.e. defined limits, consistency, verifiability, scalability, modifiability and traceability etc. [24]. Minimal means that only necessary requirements are stated so that the design space is not restricted prematurely. Focused means the impact of the requirements on the solution is clear. Completeness is a quality parameter where a
requirement specification of the complete system has to be produced and base lined before any design and implementation activities. Moreover, in the vast majority of requirement specifications, requirements are stated informally in natural language. Thus unambiguity requires the specification to be as formal as possible. The term correctness is used for adequacy. Adequacy means the quality of being able to meet a need satisfactorily, which we want to express for requirements. Many times customers are unable to assess the adequacy of the requirements due to the representation of requirements that does not match the way the customers use the system. Traceability means that rationales are given that describe why the requirements are necessary and how it is refined into sub characteristics. Scalability is a parameter that defines how much quality is required [25].

Quality requirements are critical to the success of any application. It is generally accepted that many systems fail because they don’t satisfy the requirements of the users. Since project failure leads in tremendous loss of cost, time and effort, there is a crucial need for the establishment of a requirement analysis infrastructure that promotes user involvement and ensures requirement veracity at low expenditures. One of the most difficult problems in deciding on the system requirements is the communication gap between the different stakeholders. The system analysts tend to speak with technical terms while the customer and the end-user prefer to use the language used in their daily business. As a result the analysts may not have a clear idea what kind of system they are building and the customers and the end-users are often surprised when they
notice that the final product does not fulfill their expectations and solve the problems they needed the system for in the first place[26].

2.4.2 Role of Requirement Elicitation
 Requirement elicitation involves sifting through large amounts of information and deciding what exactly is relevant. Good communication skills, both oral and written, are essential for requirement elicitation, since nearly all methods of fact finding depend on communication with clients and users. One of the most successful ways of eliciting requirements is to use Scenarios. Scenarios were the first used in the field of human-computer interaction [27].

A scenario is a description of a person’s interaction with a system. Scenarios help focus design efforts on the user’s requirements, which are distinct from technical or business requirements. A scenario is a description in natural language of a particular sequence of interactions between the user and the system. They are defined as narrative descriptions in a specific context bound in time or as specific instances containing descriptions of the environment, the context, the actors, and the actions with definite beginning and end points. They are also presented as specific instances of use cases where a scenario describes a path of actions through a use case [26]. Scenarios are concerned with the behavior of the system that is visible externally. They aim to establish what the system does from the user’s point of view [28].
2.4.3 Use of scenarios in Requirement Elicitation

- **Detailed specifications:** Scenario is a sequence of user-system interactions representing a system function from a user’s perspective. Scenarios provide a decomposition of a system into function from user’s point of view and each function can be engineered separately.

- **Accepting user’s perspective:** Scenario always view a system from the user’s point of view. This helps in validating the adequacy of requirements which is an advantage.

- **Ease in system testing:** The interaction sequences of scenarios act as a base for defining system test. Test cases can be directly derived from the scenarios resulting in the enhancement of verifiability of requirements [29].

- **Understandability:** Scenarios lead to ease of understanding the user-system interaction which leads to a natural way of understanding and discussing requirements for users as well as requirement engineers. This results in simplifying both elicitation and validation of requirements.

2.4.4 Representing the Scenarios

Scenarios are a medium for reasoning about user activities; they are used for requirement capture and analysis, user-designer communication, usability evaluation etc. Not much work has been devoted to the systematic representation of scenarios. A scenario consists of a goal (which may or may not be achieved) and a
sequence of actions that lead to it. The goal must be well defined, in such a way that it is clear whether or not it is successfully achieved. The sequence of actions leading to the goals must have definite start and end points and be expressed in a way that is as simple and concrete as possible.

The basic advantage of using a scenario is that they are particularly attractive to users, since their narrative form makes them a very effective means of communication. During requirement elicitation, scenarios can help users to explain what they do to carry out particular tasks, what information they need and what the output will be. The fact that scenarios are readily accessible by users is also a benefit to the developers, since any technique that promotes a shared understanding of requirements is valuable in system development.

Below given is the description (goals and constraints) for the sample application:

System should support the **ATM** technology, where user can themselves withdraw money, check their account balance, change their ATM card pin code etc. ATM system should act as self-service terminal for money transactions. User identifies himself with personal ATM/Debit card. Every card has a pin code which should be entered correctly. This pin code opens the security options for the respective user and allows him to access his account. User may not be able to withdraw amount if the required amount exceeds the defined limit. In another case where the account balance will be zero, system may not produce any amount.

Usually, plain narrative text is used when scenarios are defined. Scenarios of this type are convenient for reading and
writing as well, but they are typically faced with serious quality problems as they are very imprecise (Fig. 2.2). There is no difference between user actions and system responses, thus making it difficult to draw a borderline between system and its environment. For example, in the scenario of Fig.2.2, it is not clear whether the production of receipt is the last step in the scenario or is just mentioned last. No behavior is specified.

Cockburn has proposed a step-by-step description in natural language (Fig. 2.3). They exhibit a clear sequence of actions and separate normal cases from exceptional ones [30]. However non-

<table>
<thead>
<tr>
<th>Type scenario: ATM</th>
<th>Type scenario: ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>When a user wants to withdraw money from ATM, he first scans the ATM card and enters the pin code of the ATM/debit card. If the pin code is correct, system turns off the safely label allows the user to move ahead. User selects the account option. If there is required balance in the account and the amount to be withdrawn is within the specified limits, then the system produces the required amount. System finally produces the receipt asks if the user wants to continue further. If not, procedure is terminated by pressing exit.</td>
<td>Actor: User</td>
</tr>
<tr>
<td>Normal flow</td>
<td></td>
</tr>
<tr>
<td>1. Scans the ATM/Debit card.</td>
<td></td>
</tr>
<tr>
<td>2. Enters the pin code and validates the card.</td>
<td></td>
</tr>
<tr>
<td>3. Selects the account option (saving/current).</td>
<td></td>
</tr>
<tr>
<td>4. Selects withdraw option to withdraw required amount. Enters the desired amount to be withdrawn.</td>
<td></td>
</tr>
<tr>
<td>5. If user wants to withdraw more money, repeat steps 3-5.</td>
<td></td>
</tr>
<tr>
<td>6. When finished, prints receipt.</td>
<td></td>
</tr>
<tr>
<td>Alternative flow</td>
<td></td>
</tr>
<tr>
<td>1. Card is invalid : terminate.</td>
<td></td>
</tr>
<tr>
<td>3. Amount exceeds the limits : repeat process.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2 A narrative scenario

Figure 2.3 A step-by-step scenario
linear flow of actions is not systematically treated and there is no clear separation between user actions and system responses. For example, in step 2, it is not clear that who checks the pin code, user or the system.

---

Type scenario: ATM
Actor: User

Normal flow:
1. User scans his ATM card
   (System validates the card; returns the card; displays “pin code” label;)
2. User enters the pin code
   (System checks the pin code; unlocks safety labels; displays account Options)
3. User selects account option (saving/current)
   (System checks the type of account; displays options)
4. User presses ‘withdraw’ option
   (System asks for the amount to be withdrawn; checks the balance)
5. User enters the desired amount to be withdrawn
   (System produces the desired amount)
6. User presses ‘withdraw’ option again
   (System displays ‘withdraw’ dialog; go to step 3)
7. User presses ‘Exit’ key
   (System prints the receipt; displays exit message; terminate)

Alternative flows
1. System validates for the pin code, if pin code is invalid, system displays ‘Invalid’ message;
   terminate; endif
2. If user has zero balance in the account, system displays ‘zero balance’ message;
   terminate; endif
3. If amount entered exceeds the limits, system displays ‘Denied’ message;
   go to step 5; endif

---

Figure 2.4: Structured scenario

The next figure i.e. Fig. 2.4 represents a style for the representation of scenarios which is a combination of state chart based structures and structured textual representation. The distinctive features of this representation are:
• Clear separation of the events produced by an actor,
• Simple structuring constructs (if, go to, terminate) to make flow unambiguous,
• Possibility to transform this representation into state chart easily,
• Easily understandable by the users and the system as well.

2.5 Scenario Functionality Model
A system is always developed for a purpose: to provide functionality or behavior, that will satisfy the needs and wishes of clients and users. Functionality is modeled in Object Oriented system development, from the early stages of requirement engineering through diagrams that can be used as the basis for writing code. Every system is structured in a way that it has various options for the users to perform queries, functions and transactions. State charts provide powerful mechanism for structuring and abstraction [31].

2.5.1 Proposed Functionality Model
Every structure that represents a scenario correspondingly has one starting state and one normal exit state. This functionality model represents the structures of all scenarios describing interaction between users and the system. The functionality of a scenario modeled in Fig. 2.5, describes the interactions between users and system. Considering the example of ATM defined earlier, the first step in the structure that specifies to enter the system defines the first step where user has to login in the system by scanning the card.
and entering the correct pin code. Then the user can use the system to perform queries like if user wants to know the account balance (checking deposit) etc. Also various functions and transactions can be performed using the system. These scenarios can be repeated as many number of times as required by the user. The timeout transition is an exceptional behavior i.e. if a user does not completes its functions or queries or transactions, the timeout lets the system to return into a defined state.

Figure 2.5: Scenario functionality model
2.5.2 Analysis of quality parameters with functionality model:
Developing an application through creation and elaboration of scenarios is an evolutionary process in which analysis and experimentation with individual scenarios reveals increasingly more about software requirements. A development process that relies heavily on scenarios must be very sensitive to issues of completeness. The notion of completeness leads to waterfall – like process model, where specifications of the complete system have to be produced and base lined prior to any design and implementation activities. However, customers don’t always know and understand what they want. Systems and requirements evolve. So it is almost impossible to produce and freeze a complete requirement specification. Every scenario represents a partial specification that is coherent from a user’s perspective referred as partial completeness.

Two important questions are elicited here: how many scenarios are sufficient as a requirement specification from a system and is it necessary that the scenario set covers all the intended functionality. Through this model, it is found that the key scenarios provide a good specification for systems under design. Another critical issue for scenario based methods is maintaining consistency across scenarios. An explicit integration of scenario description and their implementations can promote consistency in the overall design. On the contrary, viewing every scenario as a separate entity can lead to severe inconsistency problems. Thus in scenario based approach; explicit effort is needed to ensure consistency.

The proposed model provides a framework for organizing partial specifications, keeping them consistent. Adequacy is termed
as the most important quality attribute. In many projects, customers are unable to assess the adequacy of the requirements because the way that the requirements are represented does not match the way that customers use a system and think about it. Moreover when customers don’t fully know and understand what they want, the assessment of adequacy becomes even more difficult. The above defined scenario functionality model situate requirements in the environment where a system will be used and describe them in a user-oriented way.

Together with the decomposability into user functions or transactions and the ease of understanding, the model allows for the continuous validation of written requirements against the customer’s intensions, thus yielding adequacy specifications. With this model, we can assess the adequacy of scenarios in its context and verify that the specifications express required properties of a system properly. Unambiguity, is another important quality factor, that requires the specification to be as formal as possible. However, in the vast majority of requirement specification, requirements are stated informally with natural language. Thus, unambiguity is very difficult to achieve. Using scenarios makes processes possible that use short cycles between writing and validating requirements and that define test cases derived from the scenarios early. Such processes, together with the user-orientation of scenarios, yield powerful capabilities for detecting and resolving ambiguities. Thus, scenarios do not lead to specifications that are unambiguous, but they do support processes with close feedback loops which are the natural means of detecting and resolving ambiguities in communication between humans.
For designers modeling an application in a specific scenario, one design issue concerns the attributes or the behavior this application may share with objects in other scenarios. The functionality model of scenarios provides us with validation and verification capabilities that go beyond those available for a set of isolated scenarios.

The presented model seeks to support a process in which designers experiment early with software abstraction, but always in the context of specific task requirements. This motivates and rationalizes a software design in which these requirements are easily identifiable and can be easily manipulated or modified. This is a common sort of dependency between user tasks and the system.

2.6 Conclusion:
Since inadequate, inconsistent and ambiguous requirements have a critical impact on the resulting software and may lead to failure, these requirements need to be engineered and should have continued revisions and reviews. Different types and levels of requirements must be understood to do a good job of requirements engineering. It requires an understanding of the benefits of having good requirements so that adequate resources and time are dedicated to the requirements engineering process throughout the software development life cycle. Doing requirements engineering correctly requires an interdisciplinary approach that considers the needs of multiple stakeholder groups. It also requires expertise in the various skills of requirements engineering including requirements elicitation, fusion, requirements analysis, requirements specification and requirements validation.
To accomplish this, a set of requirements engineering tasks are conducted. The repeated evaluation of requirements in the requirements engineering framework results in the achievement of those requirements which are actually intended by the stakeholders. The newly introduced phase of requirement fusion in the proposed framework will further support the requirement engineering process to achieve quality requirements for a high quality product. After introducing this framework, a new model is presented for eliciting efficient requirements that give a paradigm of requirements quality that concentrates on completeness, consistency, modifiability, verifiability etc. Requirement Engineering with scenarios supports the paradigm and potentially improves overall quality by using a systematic approach.
2.7 References


81


