3 Software Requirements : Analysis and Specifications

When we receive a request for a new software project from the customer, first of all, we would like to understand the project. The new project may replace the existing system such as preparation of students semester results electronically rather than manually. Sometimes, the new project is an enhancement or extension of a current (manual or automated) system. For example, a web enabled student result declaration system that would enhance the capabilities of the current result declaration system. No matter, whether its functionality is old or new, each project has a purpose, usually expressed in what the system can do. Hence, goal is to understand the requirements of the customer and document them properly. A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfil the system's purpose.

The hardest part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later [BROO95]. Throughout software industry's history, we have struggled with this truth. Defining and applying good, complete requirements is hard to work, and success in this endeavor has eluded many of us. Yet, we continue to make progress.

3.1 REQUIREMENTS ENGINEERING

Requirements describe the "what" of a system, not the "how". Requirements engineering produces one large document, written in a natural language, contains a description of what the system will do without describing how it will do. The input to requirements engineering is the problem statement prepared by the customer. The problem statement may give an overview of the existing system alongwith broad expections from the new system.

3.1.1 Crucial Process Steps

The quality of a software product is only as good as the process that creates it. Requirements engineering is one of the most crucial activity in this creation process. Without well-written requirements specifications, developers do not know what to build, customers do not know what to expect, and there is no way to validate that the built system satisfies the requirements.

Requirements engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behaviour and its associated constraints [HSIA93]. This process consists of four steps as shown in Fig. 3.1.

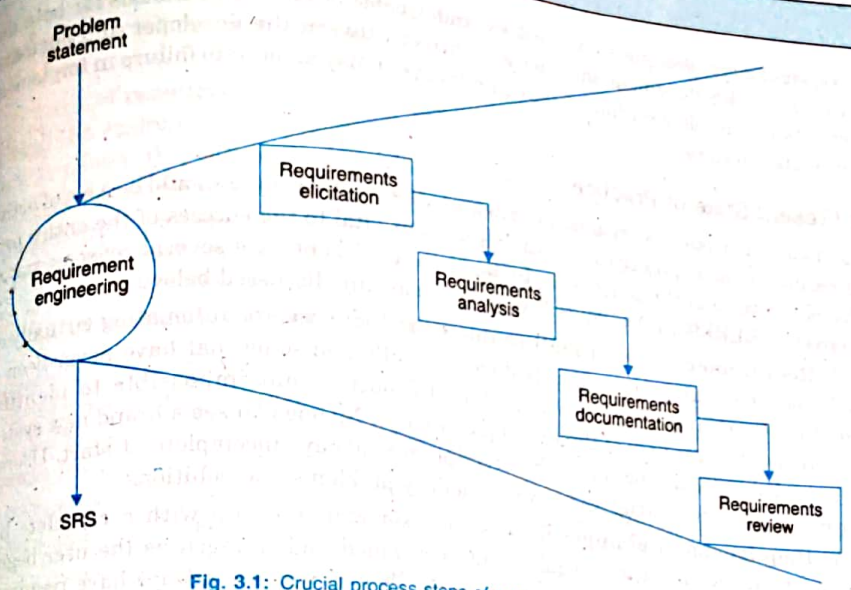


Fig. 3.1: Crucial process steps of requirement engineering

(*i*) **Requirements elicitation:** This is also known as gathering of requirements. Here, requirements are identified with the help of customer and existing systems processes, if available.

(*ii*) **Requirements analysis:** Analysis of requirements starts with requirement elicitation. The requirements are analysed in order to identify inconsistencies, defects, omissions etc. We describe requirements in terms of relationships and also resolve conflicts, if any.

(*iii*) **Requirements documentation:** This is the end product of requirements elicitation and analysis. The documentation is very important as it will be the foundation for the design of the software. The document is known as software requirements specification (SRS).

(*iv*) **Requirements review:** The review process is carried out to improve the quality of the SRS. It may also be called as requirements verification. For maximum benefits, review and verification should not be treated as a discrete activity to be done only at the end of the preparation of SRS. It should be treated as continuous activity that is incorporated into the elicitation, analysis, and documentation.

The primary output of requirements engineering is requirements specifications. If it describes both hardware and software, it is a system requirements specification. If it describes only software, it is a software requirements specification. In either case, a requirements specification must treat the system as a black box. It must delineate inputs, outputs, the functional requirements that show external behaviour in terms of input, output, and their relationships, and non-functional requirements and their constraints, including performance, portability, and reliability.

The software requirements specification (SRS) should be internally consistent; consistent with existing documents; correct and complete with respect to satisfying needs; understandable

to users, customers, designers, and testers; and capable of serving as a basis for both design and test. This SRS document may act as contract between the developer and customer. If developer fails to implement full set of requirements, it may amount to failure in implementing the contracted system.

3.1.2 Present State of Practice

Most software development organizations agree to the fact that there should be a set of activities called requirements engineering and their success is vital to the success of the entire project. So why is the state of the practice no better than it is? There are several reasons, not all of them obvious [BERR98, DAV194, HSIA93]; and some are discussed below:

1. **Requirements are difficult to uncover:** Today we are automating virtually every kind of task-some that were previously done manually and some that have never been done before. In either kind of application, it is difficult, if not impossible to identify all the requirements, regardless of the techniques we use. No one can see a brand new system in its entirety. Even if someone could, the description is always incomplete at start. Users and developers must resort to trial and error to identify problems and solutions.

2. **Requirements change:** Because no user can come up with a complete list of requirements at the outset, the requirements get added and changed as the user begins to understand the system and his or her real needs. That is why we always have requirement changes. But, project schedule is seldom adjusted to reflect these modifications. Fluid requirements make it difficult to establish a baseline from which to design and test. Finally, it is hard to justify spending resources to make a requirement specification "perfect", because it will soon change anyway. This is the biggest problem, and there is as yet no technology to overcome it. This problem is often used as an excuse to either eliminate or scale back requirements engineering effort.

3. **Over-reliance on CASE tools:** Computer Aided Software Engineering (CASE) tools are often sold as panaceas. These exaggerated claims, have created a false sense of trust, which could inflict untold damage on the Software Industry. CASE tools are as important to developers (including requirement writers) as word processors are to authors. However, we must not rely on requirements engineering tools without first understanding and establishing requirements engineering principles, techniques and processes. Furthermore, we must have realistic expectations from the tools.

4. **Tight project schedule:** Because of either lack of planning or unreasonable customer demand, many projects start with insufficient time to do a decent job. Sometimes, even the allocated time is reduced while the project is under way. It is also customary to reduce time set apart to analyze requirements, for early start of designing and coding, which frequently leads to disaster.

5. **Communication barriers:** Requirement engineering is communication intensive activity. Users and developers have different vocabularies, professional backgrounds, and tastes. Developers usually want more precise specifications while users prefer natural language. Selecting either results in misunderstanding and confusion.

6. **Market-driven software development:** Many of the software development is today market driven, developed to satisfy anonymous customers and to keep them coming back to buy upgrades.

7. **Lack of resources:** There may not be enough resources to build software that can do everything the customer wants. It is essential to rank requirements so that, in the face of pressure to release the software quickly, the most important can be implemented first.

Requirement problems are expensive and plague almost all systems and software development organizations. In most cases, the best we can hope for it is to detect errors in the requirements in time to contain them before the software is released [SAWY99]. Because of the concern for public safety, reputation and capital investments; developers began to recognize the need for clear, concise and complete requirements [COUN99].

Example 3.1: *A university wishes to develop a software system for the student result management of its M. Tech. Programme. A problem statement is to be prepared for the software development company. The problem statement may give an overview of the existing system and broad expectation from the new software system.*

Solution: The problem statement is prepared by the Examination division of the University and is given below:

"A University conducts a 4-semester M. Tech programme. The students are offered four theory papers and two Lab papers (practicals) during Ist, IInd and IIIrd semesters. The theory papers offered in these semesters are categorized as either 'Core' or 'Elective'. Core papers do not have an alternative subject, whereas elective papers may have two or more alternative subjects. Thus a student can study any subject out of the choices available for an elective paper.

In Ist, IInd and IIIrd semesters, 2 core papers and 2 elective papers are offered to each student. The students are also required to submit a term paper minor project in IInd and IIIrd semesters each. In IVth semester the students have to give a seminar and submit a dissertation on a topic/subject area of their interest.

The evaluation of each subject is done out of 100 marks. During the semester, minor exams are conducted for each semester. Students are also required to submit assignments as directed by the corresponding faculty and maintain Lab records for practicals. Based on the students' performance in minor exams, assignments, Lab records and their attendance, marks out of 40 are given in each theory paper and practical paper. These marks out of 40 account for internal evaluation of the students. At the end of each semester, major exams are conducted in each subject (theory as well as practical). These exams are evaluated out of 60 marks and account for external evaluation of the students. Thus, the total marks of a student in a subject are obtained by adding the marks obtained in internal and external evaluation.

Every subject has some credit points assigned to it. If the total marks of a student are >= 50 in a subject, he/she is considered 'Pass' in that subject otherwise the student is considered 'Fail' in that subject. If a student passes in a subject he/she earns all the credit points assigned to that subject, but if the students fails in a subject he/she does not earn any credit point in that subject. At any time, the latest information about subjects being offered in various semesters and their credit points can be obtained from University Website.

It is required to develop a system that will manage information about subjects offered in various semesters, students enrolled in various semesters, elective (s) opted by various students in different semesters, marks and credit points obtained by students in different semesters. The system should also have the ability to generate printable mark-sheets for each student. Semester-wise detailed mark-lists and student performance reports also need to be generated.

Example 3.2: *A university wishes to develop a software system for library management activities. Design the problem statement for the software company.*

Solution: The problem statement prepared by the library staff of the university is given below. Here activities are explained point wise rather than paragraph wise.

A software has to be developed for automating the manual library system of a University. The system should be standalone in nature. It should be designed to provide functionalities as explained below:

1. Issue of books

(*a*) A student of any course should be able to get books issued.

(*b*) Books from **General section** are issued to all but **Book bank** books are issued only for their respective courses.

(*c*) A limitation is imposed on the number of books a student can be issued.

(*d*) A maximum of 4 books from book bank and 3 books from general section per student is allowed.

(*e*) The books from book bank are issued for entire semester while books from general section are issued for 15 days only.

(*f*) The software takes the current system date as the date of issue and calculates the corresponding date of return.

(*g*) A bar code detector is used to save the student information as well as book information.

(*h*) The due date for return of the book is stamped on the book.

2. Return of books

(*a*) Any person can return the issued books.

(*b*) The student information is displayed using the bar code detector.

(*c*) The system displays the student details on whose name the books were issued as well as the date of issue and return of the book.

(*d*) The system operator verifies the duration for the issue and if the book is being returned after the specified due date, a fine of Re 1 is charged for each day.

(*e*) The information is saved and the corresponding updations take place in the database.

3. Query processing

(*a*) The system should be able to provide information like:

(*i*) Availability of a particular book

(*ii*) Availability of books of any particular author.

(*iii*) Number of copies available of the desired book.

(*b*) The system should be able to reserve a book for a particular student for 24 hrs if that book is not currently available.

The system should also be able to generate reports regarding the details of the books available in the library at any given time.

The corresponding printouts for each entry (issue/return) made in the system should be generated.

Security provisions like the login authenticity should be provided. Each user should have a user id and a password. Record of the users of the system should be kept in the log file. Provision should be made for full backup of the system.

3.2 TYPE OF REQUIREMENTS

There are different types of requirements such as:

(*i*) Known requirements—Something a stakeholder believes to be implemented.

(*ii*) Unknown requirements—Forgotten by the stakeholder because they are not needed right now or needed only by another stakeholder.

(*iii*) Undreamt requirements—Stakeholder may not be able to think of new requirements due to limited domain knowledge.

The term stakeholder is used to refer to any one who may have some direct or indirect influence on the system requirements. Stakeholder includes end-users who will interact with the system and every one else in an organisation who will be affected by it [SOMM01].

A known, unknown, or undreamt requirement may be functional or non-functional. Functional requirements describe what the software has to do. They are often called product features.

Non-functional requirements are mostly quality requirements that stipulate how well the software does what it has to do. Non-functional quality requirements that are especially important to users include specifications of desired performance, availability, reliability, usability and flexibility. Non-functional requirements for developers are maintainability, portability, and testability.

Some requirements are architectural, such as component-naming compatibility, interfaceability, upgradability, etc. Other requirements are constraints, such as system design constraints, standards conformance, legal issues and organisational issues. Constraints can come from users or organisations and may be functional or non-functional [ROBE02].

3.2.1 Functional and Non-functional Requirements

Software requirements are broadly classified as functional and non-functional requirements.

(*i*) **Functional requirements:** These are related to the expectations from the intended software. They describe what the software has to do. They are also called product features. Sometimes, functional requirements may also specifiy what the software should not do.

(*ii*) **Non-Functional requirements:** Non-functional requirements are mostly quality requirements that stipulate how well the software does what it has to do. Non-functional requirements that are especially important to users include specifications of desired performance, availability, reliability, usability and flexibility. Non-functional requirements for developers are maintainability, portability and testability.

The functional requirements are directly related to customer's expectations and are essential for the acceptance of product. However, non-functional requirements may make the customer happy and satisfied. These requirements are important for the success of any product. Sometimes, distinction between functional and non-functional may not be easy. If we want to develop a secure system, security becomes crucial to us. It is a non-functional requirement apparently, but when design is carried out, it may have serious implications. These implications may also be reflected in functional requirements and should therefore find a place in the functional requirements category of SRS document.

The term stakeholder is used to refer to anyone who may have some direct or indirect influence on the system requirements. "Stakeholder" includes end-users who will interact with the system and everyone else in an organisation who will be affected by it [SOMM01].

Some requirements are known requirements which a stakeholder believes to be implemented. Some are unknown to one stakeholder but may be required by another. Some requirements may be undreamt requirements for the stakeholder due to limited domain knowledge. However, a known, unknown, or undreamt requirement may be functional or non-functional depending upon its nature.

3.2.2 User and System Requirements

User requirements are written for the users and include functional and non-functional requirements. Users may not be the experts of the software field ; hence simple language should be used. The software terminologies, notations etc. should be avoided. User requirements should specify the external behaviour of the system with some constraints and quality parameters. However, design issues should be avoided. Hence, we should only highlight the overview of the system without design characteristics. System requirements are derived from user requirements. They are expanded form of user requirements. They may be used as input to the designers for the preparation of software design document. The user and system requirements are the parts of software requirements and specification (SRS) document. There are various ways of writing these requirements ; however IEEE std. 830—1998 (for SRS) framework is very popular and used by many software organisations.

3.2.3 Interface Specification

Interface issues are very important for the customers. A good software may not be appreciated if interfaces are not as per expectations of the customers. The types of interfaces are :

(*i*) **Procedural interfaces:** Some services are offered by calling interface procedures. These interfaces are called Application Programming Interfaces (APIs).

(*ii*) **Data structures:** Data structures are used to transfer information from one module to another module. This can be represented using graphical data models.

(*iii*) **Representations of data:** The issues related to ordering of bits are established here. Some are common in embedded systems and microprocessor applications. One of the methods to describe these is to use a diagram of the structure with annotations explaining the function of each group [SUMM01].

Some formal notations are available in the literature with some strengths and weaknesses.

3.3 FEASIBILITY STUDIES

Is cancellation of a project a bad news ? As per IBM report, "31% projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts" [IBMG2K]. Caper Jones reports that the average cancelled project in the US is about a year behind schedule and has consumed 200% of its expected budget by the time it is cancelled [CAPE94]. He further estimates that the work on cancelled projects comprises about 15% of the total US software efforts.

In spite of these statistics, cancelling a project is, in itself, neither good nor bad. Cancelling a project later than necessary is bad. The trick is to perform the minimum amount of work necessary to determine whether the project should be cancelled.

How do we cancel a project with the least work ? One of the most effective ways is to conduct a feasibility study to determine whether the full scale project is workable. This study culminates in a feasibility report, which enables project team, customer, or upper management make a go/no go decision about rest of the project. Feasibility studies are a time tested practice, but they are not used very much. One reason might be the term "feasibility study" itself. The term sometimes relates to technical feasibility and question of technical feasibility rarely enters in the minds of software developers. If we know that project is technically feasible, why should we conduct a feasibility study ?

For few projects, technical feasibility may be very important and significant. Is it technically feasible to provide direct communication connectivity through space from one location of globe to another location ? It is technically feasible to design a programming language using "Sanskrit" (Ancient Indian Language) ? However, for most of the projects, feasibility depends on non-technical issues like :

— Are the project's cost and schedule assumptions realistic ?

— Is the business model realistic ?

— Is there any market for the product ?

3.3.1 Purpose of Feasibility Studies

The decision to implement any new project or program must be based on a thorough analysis of the proposed project/program.

A feasibility study is defined as an evaluation or analysis of the potential impact of a proposed project or program. It is conducted to assist decision makers in determining whether or not to implement a project/program. It is based on extensive research on both the current practices and the proposed project or program and its impact on the system as a whole. It should also contain extensive data analysis related to financial and operational impact and should include advantages and disadvantages of both the current situation and the proposed project.

3.3.2 Focus of Feasibility Studies

Steve McConnel [SIEV98] has given the following points on which a feasibility study should focus and give proper emphasis to get good results:

(*i*) Is the product concept viable ?

(*ii*) Will it be possible to develop a product that matches the project's vision statement ?

(*iii*) What are the current estimated cost and schedule for the project ?

(*iv*) How big is the gap between the original cost and schedule targets and current estimates ?

(*v*) Is the business model for software justified when the current cost and schedule estimates are considered ?

(*vi*) Have the major risks to the project been identified and can they be surmounted ?

(*vii*) Are the specifications complete and stable enough to support remaining development work ?

(*viii*) Have users and developers been able to agree on a detailed user interface prototype ? If not, are the requirements really stable ?

(*ix*) Is the software development plan complete and adequate to support further development work ?

The work done during the first 10 to 20 percent of the project should sufficiently answer these questions and give the client or top management enough information to decide whether to fund the rest of the project.

Breaking a software project into a feasibility study phase and a main development phase (starting from requirements elicitation) helps software organisations in at least three ways:

(*i*) Some people view any cancelled project as a failure. But a project cancelled at 10 to 20 percent level vis-a-vis completion point should not be considered a failure. Cancelling one project that ultimately goes no where after it is only 10 to 20 percent complete instead of 80 to 90 percent complete should always be a good decision.

(*ii*) Feasibility study can be conducted with marginal funds. After the final decision about project, more accurate and realistic estimate can be prepared. This may help us to reduce the cost variations.

(*iii*) The project team will complete 10 to 20 percent of a project before requesting funding for the rest of it forces a focus on upstream activities that are critical to a project's success. Otherwise these activities are often ignored, and the damaging consequences of such neglect would not become apparent until late in the project.

3.4 REQUIREMENTS ELICITATION

Requirements elicitation is perhaps the most difficult, most critical, most error-prone, and most communication intensive aspect of software development. Elicitation can succeed only through an effective customer-developer partnership [WIEG99].

The real requirements actually reside in user's mind. Hence the most important goal of requirement engineering is to find out what users really need. Users need can be identified only if we understand the expectations of the users from the desired software.

It is the activity that helps to understand the problem to be solved. Requirements are gathered by asking questions, writing down the answers, asking other questions, etc. Hence, requirements gathering is the most communications intensive activity of software development. Developers and Users have different mind set, expertise and vocabularies. Due to communication gap, there are chances of conflicts that may lead to inconsistencies, misunderstanding and omission of requirements.

Therefore, requirements elicitation requires the collaboration of several groups of participants who have different background. On the one hand, customers and users have a solid background in their domain and have a general idea of what the software should do. However, they may have little knowledge of solfware development processes. On the other hand, the developers have experience in developing software but may have little knowledge of everyday environment of the users. Moreover each group may be using incompatible terminologies.

There are number of requirements elicitation methods and few of them are discussed in the following sections. Some people think that one methodology is applicable to all situations, however, generally speaking, one methodology cannot possibly be sufficient for all conditions [MACA96]. We select a particular methodology for the following reason(s):

(*i*) It is the only method that we know.

(*ii*) It is our favorite method for all situations.

(*iii*) We understand intuitively that the method is effective in the present circumstances.

Clearly, third reason demonstrates the most maturity and leads to improved understanding of stakeholder's needs and thus resulting system will satisfy those needs. Unfortunately, most of us do not have the insight necessary to make such an informed decision, and therefore rely on the first two reasons [HICK03].

3.4.1 Interviews

After receiving the problem statement from the customer, the first step is to arrange a meeting with the customer. During the meeting or interview, both the parties would like to understand each other. Normally specialised developers, often called 'requirement engineers' interact with the customer. The objective of conducting an interview is to understand the customer's expectations from the software. Both parties have different feelings, goals, opinions, vocabularies, understandings, but one thing is common, both want the project to be a success. With this in mind, requirement engineers normally arrange interviews. Requirement engineers must be open minded and should not approach the interview with pre-conceived notions about what is required.

Interview may be open-ended or structured. In open-ended interview, there is no pre-set agenda. Context free questions may be asked to understand the problem and to have an overview of the situation. For example, for a "result management system", requirement engineer may ask:

- Who is the controller of examination ?
- Who has requested for such a software ?
- How many officers are placed in the examination division ?
- Who will use the software ?
- Who will explain the manual system ?

- Is there any opposition for this project ?
- How many stakeholders are computer friendly ?

Such questions help to identify all stakeholders who will have interest in the software to be developed.

In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview. Interview may be started with simple questions to set people at ease. After making atmosphere comfortable and calm, specific questions may be asked to understand the requirements. The customer may be allowed to voice his or her perceptions about a possible solution.

Selection of stakeholder: It will be impossible to interview every stakeholder. Thus, representatives from groups must be selected based on their technical expertise, domain knowledge, credibility, and accessibility. There are several groups to be considered for conducting interviews:

(*i*) Entry level personnel: They may not have sufficient domain knowledge and experience, but may be very useful for fresh ideas and different views.

(*ii*) Mid-level stakeholders: They have better domain knowledge and experience of the project. They know the sensitive, complex and critical areas of the project. Hence, requirement engineers may be able to extract meaningful and useful information. Project leader should always be interviewed.

(*iii*) Managers or other Stakeholders: Higher level management officers like Vice-Presidents, General Managers, Managing Directors should also be interviewed. Their expectations may provide different but rich information for the software development.

(*iv*) Users of the software: This group is perhaps the most important because they will spend more time interacting with the software than any one else. Their information may be eye opener and may be original at times. Only caution required is that they may be biased towards existing systems.

Types of questions: Questions should be simple and short. Two or three questions rolled into one can lead to compound requirements statements that are difficult to interpret and test. It is important to prepare questions, but reading from the questionnaire or only sticking to it is not desirable. We should be open for any type of discussion and any direction of the interview. For the "result management system" we may ask:

- Are there any problems with the existing system ?
- Have you faced calculation errors in past ?
- What are the possible reasons of malfunctioning ?
- How many students are enrolled presently ?
- What are the possible benefits of computerising this system ?
- Are you satisfied with current processes and policies ?
- How are you maintaining the records of previous students ?
- What data, required by you, exists in other systems ?
- What problems do you want this system to solve ?

- Do you need additional functionality for improving the performance of the system ?
- What should be the most important goal of the proposed development ?

These questions will help to start the communication that is essential for understanding the requirements. At the end of this, we may have wide variety of expectations from the proposed software.

3.4.2 Brainstorming Sessions

Brainstorming is a group technique that may be used during requirements elicitation to understand the requirements. The group discussions may lead to new ideas quickly and help to promote creative thinking.

It is intended to generate lots of ideas, with full understanding that they may not be useful. The theory is that having a long list of requirements from which to choose is far superior to starting with a blank slate. Requirements in the long list can be categorized, prioritized, and pruned [ROBE02].

Brainstorming has become very popular and is being used by most of the companies. It promotes creative thinking, generates new ideas and provides plateform to share views, apprehensions expectations and difficulties of implementation. All participants are encouraged to say whatever ideas come to mind, whether they seem relevant or not. No one will be criticized for any idea, no matter how goofy it seems, as the responsibility of the participant is to generate views and not to vet them.

This group technique may be carried out with specialised groups like actual users, middle level managers etc., or with total stakeholders. Sometimes unnatural groups are created that may not be appriciated and are uncomfortable for participants. At times, only superficial responses may be gathered to technical questions. In order to handle such situations, a highly trained facilitator may be required. The facilitator may handle group bias and group conflicts carefully. The facilitator should also be cautious about individual egos, dominance and will be responsible for smooth conduct of brainstorming sessions. He or she will encourage the participants, ensure proper individual and group behaviour and help to capture the ideas. The facilitator will follow a published agenda and restart the creative process if it falters.

Every idea will be documented in such a way that everyone can see it. White boards, overhead transparencies or a computer projection system can be used to make it visible to every participant. After the session, a detailed report will be prepared and facilitator will review the report. Every idea will be written in simple english so that it conveys same meaning to every stakeholder. Incomplete ideas may be listed separately and should be discussed at length to make them complete ideas, if possible. Finally, a document will be prepared which will have list of requirements and their priority, if possible.

3.4.3 Facilitated Application Specification Technique

This approach is similar to brainstorming sessions and the objective is to bridge the expectation gap – a difference between what developers think they are supposed to build and what customers think they are going to get. In order to reduce expectation gap, a team oriented approach is developed for requirements gathering and is called Facilitated Application Specification Technique (FAST).

This approach encourages the creation of a joint team of customers and developers who work together to understand the expectations and propose a set of requirements. The basic guidelines for FAST are given below:

- Arrange a meeting at a neutral site for developers and customers.
- Establishment of rules for preparation and participation.
- Prepare an informal agenda that encourages free flow of ideas.
- Appoint a facilitator to control the meeting. A facilitator may be a developer, a customer, or an outside expert.
- Prepare a definition mechanism-board, flip charts, worksheets, wall stickies, etc.
- Participants should not criticize or debate.

FAST session preparations

Each FAST attendee is asked to make a list of objects that are:

(*i*) part of the environment that surrounds the system

(*ii*) produced by the system

(*iii*) used by the system.

In addition, each attendee is asked to make another list of services (processes or functions) that manipulate or interact with the objects. Finally, lists of constraints (*e.g.*, cost, size) and performance criteria (*e.g.*, speed, accuracy) are also developed. The attendees are informed that the lists are not expected to be exhaustive but are expected to reflect each person's perception of the system [PRES2K].

Activities of FAST session

The activities during FAST session may have the following steps:

- Each participant presents his or her lists of objects, services, constraints, and performance for discussion. Lists may be displayed in the meeting by using board, large sheet of paper or any other mechanism, so that they are visible to all the participants.
- The combined lists for each topic are prepared by eliminating redundant enteries and adding new ideas.
- The combined lists are again discussed and consensus lists are finalised by the facilitator.
- Once the consensus lists have been completed, the team is divided into smaller subteams, each works to develop mini-specifications for one or more enteries of the lists.
- Each subteam then presents mini-specifications to all FAST attendees. After discussion, additions or deletions are made to the lists. We may get new objects, services, constraints, or performance requirements to be added to original lists.
- During all discussions, the team may raise an issue that cannot be resolved during the meeting. An issues list is prepared so that these ideas will be considered later.
- Each attendee prepares a list of validation criteria for the product/system and presents the list to the team. A consensus list of validation criteria is then created.

- A subteam may be asked to write the complete draft specifications using all inputs from the FAST meeting.

FAST is not a panacea of the problems encountered in early requirements elicitation but it helps to understand the requirements and bridge the expectation gap of develops and customers.

3.4.4 Quality Function Deployment

It is a quality management technique that helps to incorporate the voice of the customer. The voice is then translated into technical requirements. These technical requirements are documented and results is the software requirements and specification document. These requirements are further translated into design requirements. Here, customer satisfaction is of prime concern and thus QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the software engineering process [PRES2K]. Three types of requirements are identified [ZULT92]:

(*i*) **Normal requirements:** The objectives and goals of the proposed software are discussed with the customer. If this category of requirements (normal) are present, the customer is satisfied. Examples related to result management system might be: entry of marks, calculation of results, merit list report, failed students report, etc.

(*ii*) **Expected requirements:** These requirements are implicit to the software product and may be so obvious that customer does not explicitly state them. If such requirements are not present, customer will be dissatisfied with the software. Examples of expected requirements may be: protection from unauthorised access, some warning system for wrong entry of data, the feasibility for modification of any record only by a fool proof system for the identification of person alongwith date and time of modification, etc.

(*iii*) **Exciting requirements:** Some features go beyond the customer's expectations and prove to be very satisfying when present. Examples of exciting requirements for result management system may be: if an unauthorised access is noticed by the software, it should immediately shutdown all the processes and an E-mail is generated to the system administrator, an additional copy of important files is maintained and may be accessed by system administrator only, sophisticated virus protection system etc.

The QFD method has the following steps [ROBE02]:

(*i*) Identify all the stakeholders *e.g.*, customers, users, and developers. Also identify any initial constraints identified by the customer that affect requirements development.

(*ii*) List out requirements from customer ; inputs, considering different viewpoints. Requirements are expression of what the system will do, which is both perceptible and of value to customers. Some customer's expectations may be unrealistic or ambiguous and may be translated into realistic or unambiguous requirements if possible.

(*iii*) A value indicating a degree of importance, is assigned to each requirement. Thus, customer determines the importance of each requirement on a scale of 1 to 5 as given below:

5 points:	Very important
4 points:	important

3 points: not important, but nice to have
2 points: not important
1 point: unrealistic, requires further exploration

Stakeholders will have their own unique set of criteria for determining the 'importance' or 'value' *of a requirement. It may be based on cost/benefit analysis particular to the project.*

Requirement engineers may review the final list of requirements and categorise like:

(*i*) it is possible to achieve

(*ii*) it should be deferred and the reason thereof

(*iii*) it is impossible and should be dropped from consideration.

The first category requirements will be implemented as per priority (Importance value) assigned with every requirement. If time and effort permits, second category requirements may be reviewed and few of them may be transferred to category first for implementation.

3.4.5 The Use case Approach

For many years, requirement engineers have used stories or scenarios to explain the interaction of a user with the proposed software system in order to gather the requirements. More recently, Ivar Jacobson and others [JACO99] formalised this into use case approach to requirements elicitation and modeling. Initially, use cases were designed for object oriented software development world, however, they can be applied to any project that follow any development approach because the user does not care how we develop the software. The focus on what the users need to do with the system is much more powerful than the traditional elicitation approach of asking users what they want the system to do [WIEG99].

This approach uses a combination of text and pictures in order to improve the understanding of requirements. The Use cases describe what of a system and not 'how'. They only give functional view of the system.

The terms use case, use case scenario, and use case diagram are often interchanged, but in fact they are different. Use cases are structured outline or templates for the description of user requirements, modeled in a structured language like english. Use case scenarios are unstructured descriptions of user requirements. Use case diagrams are graphical representations that may be decomposed into further levels of abstraction. The following components are used for the design of the use case approach.

Actor: An actor or external agent, lies outside the system model, but interacts with it in some way. An actor may be a person, machine, or an information system that is external to the system model. An actor is represented as stick figure and is not part of the system itself. Customers, users, external devices, or any external entity interacting with the system are treated as actors.

We should not confuse the actors with the devices they use. Devices are typically mechanisms that actors use to communicate with the system, but they are not actors themselves. We are writing this book on a computer, but the keyboad is not the user of the word processing program; but we are. Other devices, such as disk drives, tape drives, or communication equipment including printers have no place in use case diagram, although they are important

to the design of the system. The purpose of devices is to support some required behaviour of the system, but devices do not define the requirements of the system. Often systems must produce a printed report of information that it contains. We may want to show printer as an actor that then forwards the report to the real actor. This is not correct. Printer is not an actor; it is just a mechanism for conveying information [BITT03].

Cockburn [COCK01] distinguishes between primary and secondary actors. A primary actor is one having a goal requiring the assistance of the system. A secondary actor is one from which the system needs assistance.

Use cases: A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the services that satisfies the goal. It also includes possible variants of this sequence, *e.g.*, alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behaviour, error handling etc. The system is treated as a 'black box', and the interactions with the system, including responses, are as perceived from outside the system.

Thus, use cases capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore, defines all behaviour required of the system, bounding the scope of the system.

Use cases are written in an easy to understand structured narrative—the vocabulary of the domain. The users may validate the use cases and may involve in the process of gathering and defining the requirements [MALA01].

There is no standard use case template for writing use cases. The Jacobson et al. [JACO99] proposed a template for writing use cases and is given in Table 3.1(*a*). This template captures requirements in an effective way and is therefore becoming popular. Another similar template is also given in Table 3.1(*b*) which is also used by many organisations.

Use case guidelines

The following provides an outline of a process for creating use cases:

- Identify all the different users of the system.
- Create a user profile for each category of users, including all the roles the users play that are relevant to the system. For each role, identify all the significant goals the users have that the system will support. A statement of the system's value proposition is useful in identifying significant goals.
- Create a use case for each goal, following the use case template. Maintain the same level of abstraction throughout the use case. Steps in higher level use cases may be treated as goals for lower level (*i.e.*, more detailed), sub-use cases.
- Structure the use cases. Avoid over-structuring, as this can make the use cases harder to follow.
- Review and validate with users.

Table 3.1(a): Use case template

1.	**Brief Description.** Describe a quick background of the use case.
2.	**Actors.** List the actors that interact and participate in this use case.
3.	**Flow of Events.** 3.1. ***Basic flow.*** List the primary events that will occur when this use case is executed. 3.2. ***Alternative flows.*** Any subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow. A use case can have as many alternative flows as required.
4.	**Special Requirements.** Business rules for the basic and alternative flows should be listed as special requirements in the use case narration. These business rules will also be used for writing test cases. Both success and failure scenarios should be described here.
5.	**Pre-conditions.** Pre-conditions that need to be satisfied for the use case to perform.
6.	**Post-conditions.** Define the different states in which you expect the system to be in, after the use case executes.
7.	**Extension Points.**

Table 3.1(b): Use case template

1.	**Introduction.** Describe brief purpose of the use case.
2.	**Actors.** List the actors that interact and participate in this use case.
3.	**Pre-condition.** Condition that need to be satisfied for the use case to execute.
4.	**Post-condition.** After the execution of the use case, different states of the systems are defined here.
5.	**Flow of Events.** 5.1. ***Basic flow.*** List the primary events that will occur when this use case is executed. 5.2. ***Alternate flow.*** Any other possible flow in this use case, if there, should be separately listed. A use case may have many alternate flows.
6.	**Special Requirements.** Business rules for the basic and alternate flows should be listed as special requirements. Both success and failure scenarios should be described.
7.	**Related use cases.** List the related use cases, if any.

Use case diagrams

A use case diagram visually represents what happens when an actor interacts with a system. Hence, a use case diagram captures the functional aspects of a system. The system is shown as a rectangle with the name of the system (or subsystem) inside, the actors are shown as stick figures (even the non human ones), the use cases are shown as solid bordered ovals labeled with the name of the use case, and relationships are lines or arrows between actors and use cases and/or between the use cases themselves. These components are given in Fig. 3.2.

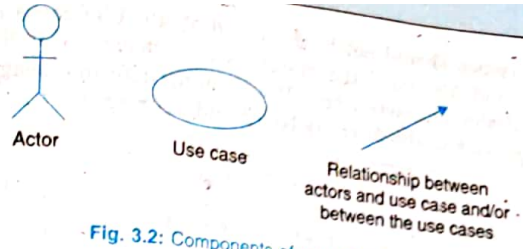


Fig. 3.2: Components of use case diagram

Actors appear outside of the rectangle since they are external to the system. Use cases appear within the rectangle, providing functionality. A relationship or association is a solid line between an actor and each use case in which actor participates—the involvement can be any kind, not necessarily one of the actor initiating the use case functionality.

Fig. 3.2 shows an example of a use case diagram whose "Problem statement" is given in Table 3.2.

Table 3.2: Problem statement for railway reservation system

PROBLEM STATEMENT FOR RAILWAY RESERVATION SYSTEM

A software has to be developed for automating the manual railway reservation system. The system should be distributed in nature. It should be designed to provide functionalities as explained below:

1. **Reserve Seat:** A passenger should be able to reserve seats in the train. A reservation form is filled by the passenger and given to the clerk, who then checks for the availability of seats for the specified date of journey. If seats are available, then the entries are made in the system regarding the train name, train number, date of journey, boarding station, destination, person name, sex and total fare. Passenger is asked to pay the required fare and the tickets are printed. It the seats are not available then the passenger is informed.
2. **Cancel Reservation:** A passenger wishing to cancel a reservation is required to fill a form. The passenger then submits the form and the ticket to the clerk. The clerk then deletes the entries in the system and changes in the reservation status of that train. The clerk crosses the ticket by hand to mark as cancelled.
3. **Update Train Information:** Only the administrator enters any changes related to the train information like change in the train name, train number, train route etc. in the system.
4. **Report Generation:** Provision for generation of different reports should be given in the system. The system should be able to generate reservation chart, monthly train report etc.
5. **Login:** For security reasons all the users of the system are given a user *id* and a password. Only if the *id* and password are correct the user is allowed to enter the system.
6. **View Reservation Status:** All the users should be able to see the reservation status of the train online. The user needs to enter the train number and the pin number printed on his ticket so that the system can display his current reservation status like confirmed, RAC or Wait listed.
7. **View Train Schedule:** Provision should be given to see information related to the train schedules for the entire train network. The user should be able to see the train name, train number, boarding and destination stations, duration of journey etc.

Use cases should not be used to capture all the details of system. The granularity to which we define use cases in a diagram should be enough to keep the use case diagram uncluttered and readable, yet, be complete without missing significant aspects of the required

functionality. Design issues should not be discussed at all. Use cases are meant to capture "what" the system is, and not "how" the system will be designed or built. Hence use cases should be free of any design characteristics. If we end up defining design characteristics in a use case, we need to go back to the drawing board and start again.

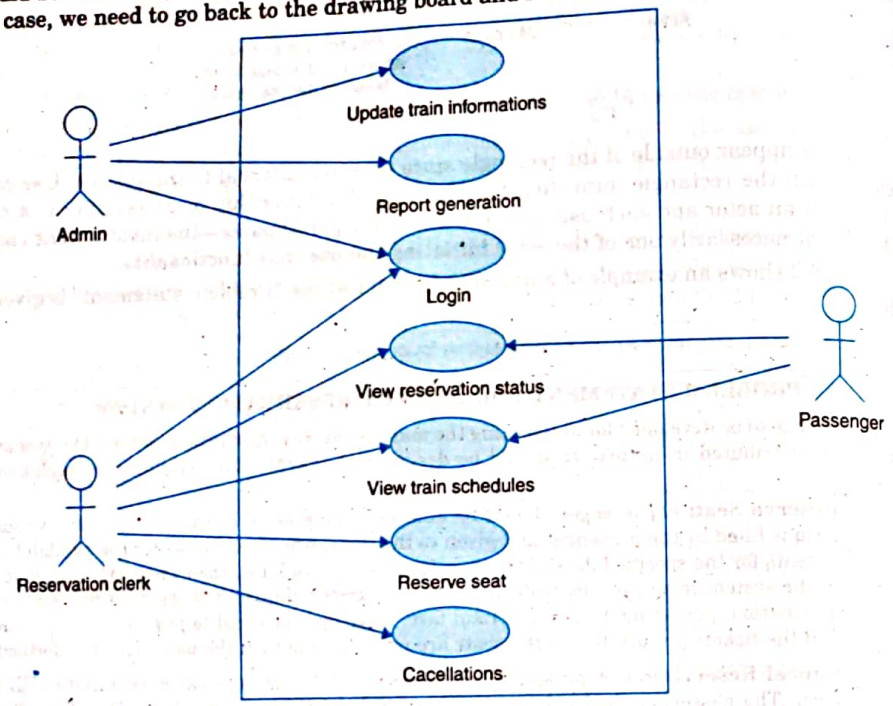


Fig. 3.3: Use case diagram for railway reservation system

3.5 REQUIREMENTS ANALYSIS

Requirements analysis is very important and essential activity after elicitation. We analyze, refine and scrutinize the gathered requirements in order to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the

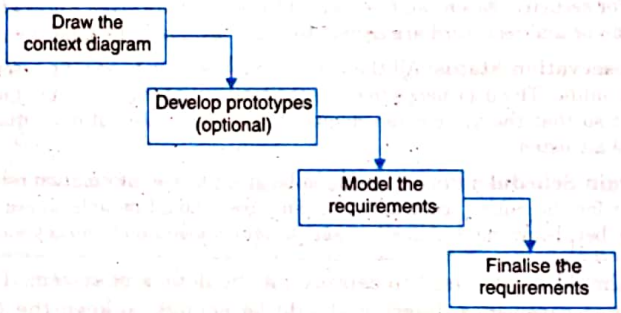
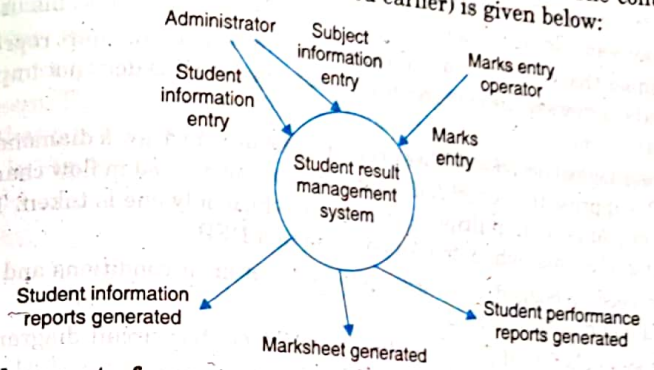


Fig. 3.4: Requirements analysis steps

entire system. After the completion of analysis, it is expected that the understandability of the project may improve significantly. Here, we may also interact with the customer to clarify points of confusion and to understand which requirements are more important than others. The various steps of requirements analysis are shown in Fig. 3.4.

(*i*) **Draw the context diagram:** The context diagram is a simple model that defines the boundaries and interfaces of the proposed system with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system (as discussed earlier) is given below:



(*ii*) **Development of a prototype (optional):** One effective way to find out what the customer really wants is to construct a prototype, something that looks and preferably acts like a part of the system they say they want.

We can use their feedback to continuously modify the prototype until the customer is satisfied. Hence, prototype helps the client to visualise the proposed system and increase the understanding of requirements. When developers and users are not certain about some of the requirements, a prototype may help both the parties to take a final decision.

Some projects are developed for general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though, persons who try out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others. Some projects are developed for a specific customer under contract. On such projects, only that customer's opinion counts, so the prototype should be shown to the prospective users in the customer organisation.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity. Although many organisations are developing prototypes for better understanding before the finalisation of SRS.

(*iii*) **Model the requirements:** This process usually consists of various graphical representations of the functions, data entities, external entities and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing and superfluous requirements. Such models include data flow diagrams, entity relationship diagrams, data dictionaries, state-transition diagrams etc.

(*iv*) **Finalise the requirements:** After modeling the requirements, we will have better understanding of the system behaviour. The inconsistencies and ambiguties have been identified and corrected. Flow of data amongest various modules has been analysed. Elicitation and analysis activities have provided better insight to the system. Now we finalise the analysed requirements and next step is to document these requirements in a prescribed format.

3.5.1 Data Flow Diagrams

Data flow diagrams (DFD) are used widely for modeling the requirements. They have been used for many years prior to the advent of computers. DFDs show the flow of data through a system. The system may be a company, an organisation, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or a bubble chart.

The following observations about DFDs are important [DAV190]:

1. All names should be unique. This makes it easier to refer to items in the DFD.
2. Remember that a DFD is not a flow chart. Arrows in a flow chart represent the order of events; arrows in DFD represent flowing data. A DFD does not imply any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represent decision points with multiple exit paths of which only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in Fig. 3.5 [SAGE90].

Symbol	Name	Function
	Data Flow	Used to connect processes to each other, to sources or sinks; the arrowhead indicates direction of data flow.
	Process	Performs some transformation of input data to yield output data.
	Source or Sink (External entity)	A source of system inputs or sink of system outputs.
	Data store	A repository of data; the arrowheads indicate net inputs and net outputs to store.

Fig. 3.5: Symbols for data flow diagrams

A circle (bubble) shows a process that transforms data inputs into data outputs. A curved line shows flow of data into or out of a process or data store. A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have element or group of elements. Source or sink is an external entity and acts as a source of system inputs or sink of system outputs.

Leveling

The DFD may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. A level-0 DFD, also called a fundamental system model or *context diagram* represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively [PRES2K]. Then the system is decomposed and represented as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the problem at hand is well understood. It is important to preserve the number of inputs and outputs between levels; this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs, x_1 and x_2, and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in Fig. 3.6 [DEMA79, DAV190].

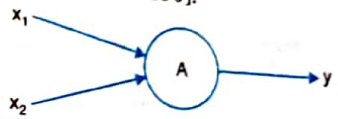


Fig. 3.6: Level-0 DFD

The level-0 DFD, also called context diagram of result management system is shown in Fig. 3.7. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flows may also need to be decomposed. Level-1 DFD of result management system is given in Fig. 3.8.

This provides a detailed view of requirements and flow of data from one bubble to the another.

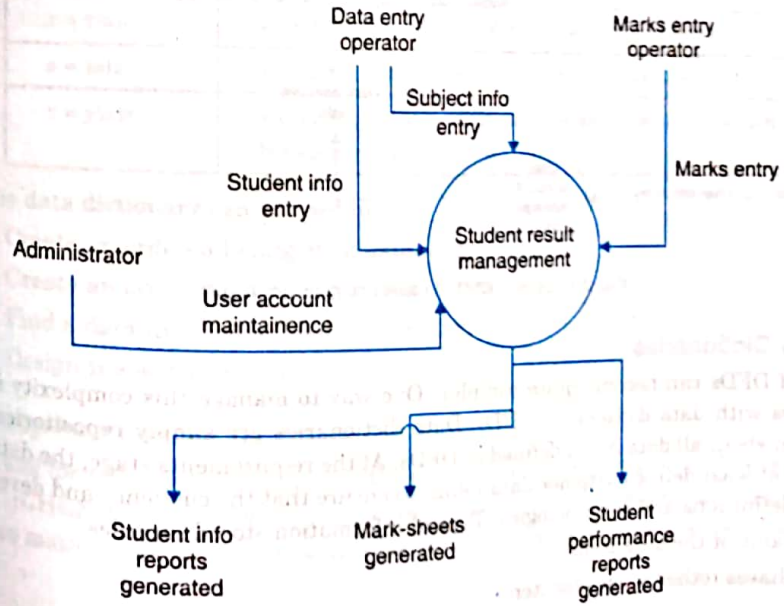


Fig. 3.7: 0-Level DFD or context diagram of result management system

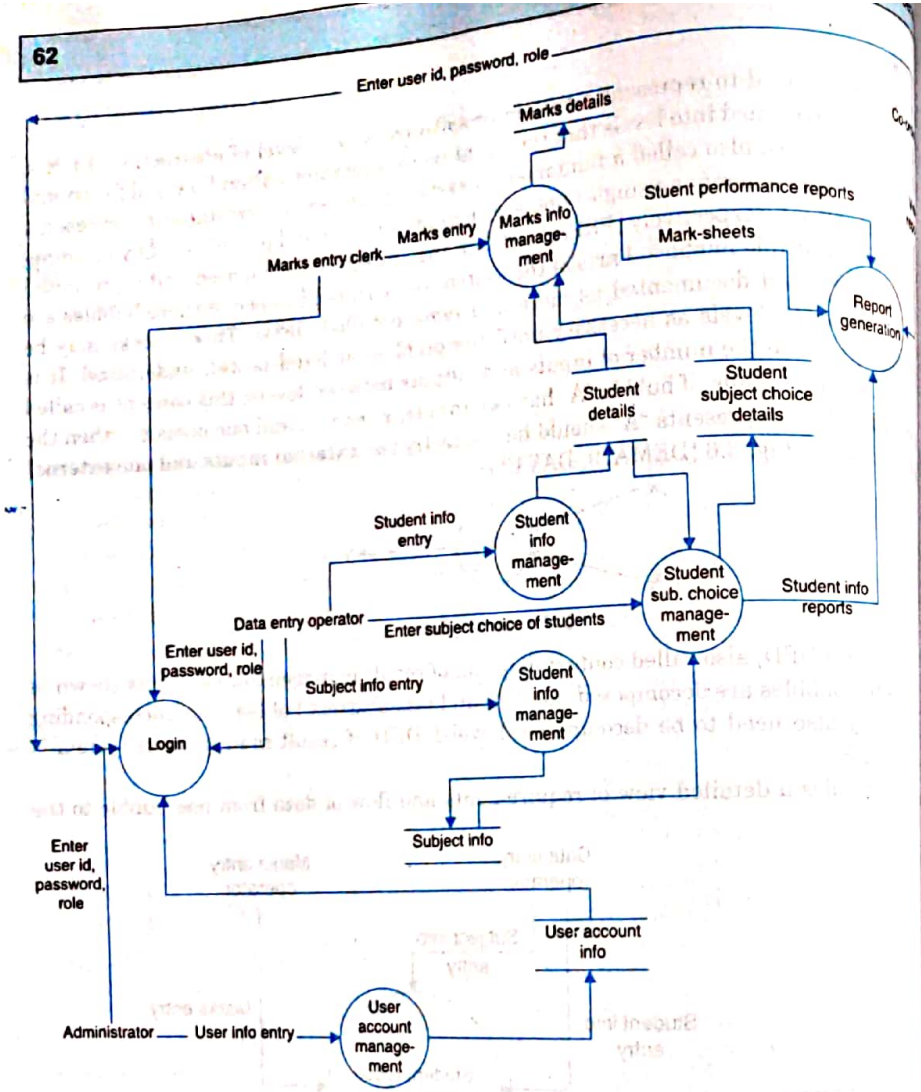


Fig. 3.8: Level-1 DFD of result management system

3.5.2 Data Dictionaries

Families of DFDs can become quite complex. One way to manage this complexity is to ment DFDs with data dictionaries (DD). Data dictionaries are simply repositories to information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developer the same definitions and terminologies. Typical information stored includes:

- Name of the data item
- Aliases (other names for item)

- Description/purpose
- Related data items
- Range of values
- Data structure definition/form

The name of the data item is self-explanatory. Aliases include other names by which this data item is called *e.g.*, DEO for Data Entry Operator and DR for deputy Registrar. Description/Purpose is a textual description of what the data item is used for or why it exists. Related data items capture relationships between data items *e.g.*, total_marks must always equal to internal_marks plus external_marks.

Range of values records all possible values, *e.g.*, total marks must be positive and between 0 to 100. Data flows capture the names of the processes that generate or receive the data item. If data item is primitive, then data structure definition/form captures the physical structure of the data item. If the data is itself a data aggregate, then data structure definition/form captures the composition of the data items in terms of other data items [DAV190]. The mathematical operators used within the data dictionary are defined in Table 3.3 [DEMA79].

Table 3.3: Data dictionary notation and mathematical operators

Notation	*Meaning*
$x = a + b$	x consists of data elements a and b
$x = [a/b]$	x consists of either data element a or b
$x = a$	x consists of an optional data element a
$x = y\{a\}$	x consists of y or more occurrences of data element a
$x = \{a\}z$	x consists of z or fewer occurrences of data element a
$x = y\{a\}z$	x consists of some occurrences of data element a which are between y and z.

The data dictionary can be used to:

- Create an ordered listing of all data items.
- Create an ordered listing of a subset of data items.
- Find a data item name from a description.
- Design the software and test cases.

3.5.3 Entity-Relationship Diagrams

Another tool for requirement analysis is the entity-relationship diagram, often called as "E-R diagram" [CHEN76]. It is a detailed logical representation of the data for an organisation and uses three main constructs *i.e.*, data entities, relationships, and their associated attributes.

Entities

An entity is a fundamental thing of an organisation about which data may be maintained. A entity has its own identity, which distinguishes it from each other entity. An entity type is the description of all entities to which a common definition and common relationships and attributes apply.

Consider a university that offers both regular and distance education programmes. These programmes are offered to national and international students.

PROGRAMME and STUDENT are both entity types in this example. Regular and distance education are entities of PROGRAMME whereas national and international are entities of STUDENT.

We use capital letters in naming an entity type and in an ER diagram the name is placed inside a rectangle representing that entity as shown in Fig. 3.9.

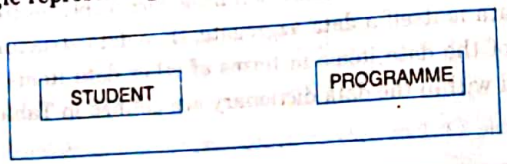


Fig. 3.9: Two entity types in an E-R diagram

Relationships

A relationship is a reason for associating two entity types. These relationships are sometimes called binary relationships because they involve two entity types. Some forms of data models allow more than two entity types to be associated. A STUDENT is registered for a PROGRAMME. Relationships are represented by diamond notation in the E-R diagram as shown in Fig. 3.10.

Fig. 3.10: Relationships added to ERD

We consider another example in which, a teaching department of a university is interested in tracking which subjects each of its students has completed. This leads to a relationship called "completes" between the STUDENT and SUBJECT entity types. This is shown in Fig. 3.11.

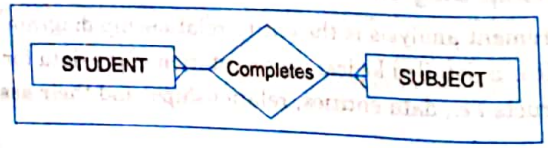


Fig. 3.11: Relationships in ERD

As indicated by the arrows, this is a many-to-many relationship. Each student may complete more than one subject, and more than one student may complete each subject.

Degree of relationships

The degree of a relationship is the number of entity types that participate in that relationship. Thus, relationship "completes" shown in Fig. 3.11 is of degree two, since there are two entity types: STUDENT and SUBJECT. The three most common relationships in E-R models are unary (degree 1), binary (degree 2), and ternary (degree three). Higher-degree relationships are possible, but they are rarely encountered in practice.

Unary relationship

This is also called recursive relationship. It is a relationship between the instances of one entity type. An instance is a single occurrence of an entity type. There may be many instances of an entity type. For example, there is one STUDENT entity type in universities, but there may be hundreds of instances of this entity type in the database. In Fig. 3.12, Is-Married-to is shown as one to one relationship between instances of the PERSON entity type. That is, each person may be currently married to one other person. In the second example, "Is friend of" is shown as one to many relationships between instances of the STUDENT entity type.

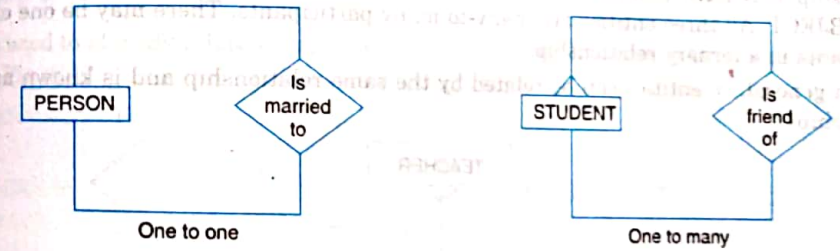


Fig. 3.12: Unary relationships

Binary relationship

It is a relationship between instances of two entity types and is the most common type of relationship encountered in E-R diagrams. Fig. 3.13 shows three examples.

The first (one to one) indicates that a STUDENT is assigned a STUDENT-ID, and each STUDENT-ID is assigned to a STUDENT. The second (one to many) indicates that a PROGRAMME may have many students, and each STUDENT belongs to only one PROGRAMME.

The third (many to many) shows that a STUDENT may register for more than one SUBJECT, and that each SUBJECT may have many STUDENT registrants.

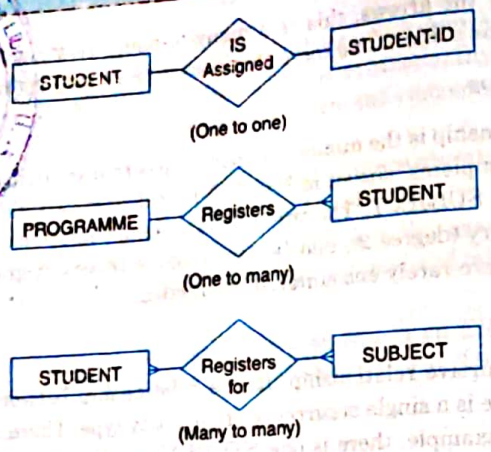


Fig. 3.13: Binary relationships

Ternary relationships

It is a simultaneous relationship amongst instances of three entity types. In Fig. 3.14, the relationship 'may have' provides the association of three entities *i.e.*, TEACHER, STUDENT and SUBJECT. All three entities are many-to many participants. There may be one or many participants in a ternary relationship.

In general, '*n*' entities can be related by the same relationship and is known as *n*-ary relationship.

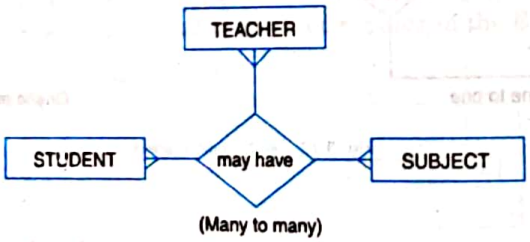


Fig. 3.14: Ternary relationship

Cardinalities and optionality

Suppose that there are two entity types, A and B, that are connected by a relationship. The cardinality of a relationship is the number of instances of entity B that can be associated with each instance of entity A.

Consider the example shown in Fig. 3.15. a student may register for many subjects.

Fig. 3.15: Use of cardinality

In the terminology, we have discussed so far, this example has "one-to many" relationship. Yet it may also be true that a subject may not have any student at specific instance of time. We need a more precise notation to indicate the range of cardinalities for a relationship.

The minimum cardinality of a relationship is the minimum number of instances of entity B that may be associated with each instance of entity A. If minimum number of students available for a subject is zero, we say that subject is an optional participant in the 'register for' relationship. When the minimum cardinality of a relationship is one, then we say entity B is a mandatory participant in the relationship. The maximum cardinality is the maximum number of instances. In our example, maximum is 'many'. The modified E-R diagram is given in Fig. 3.16. The zero through the line near the SUBJECT entity means a minimum cardinality of zero, while the crow's foot notation means a 'many' maximum cardinality.

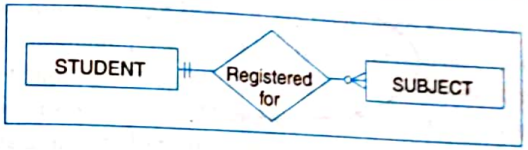


Fig. 3.16: Modified ER diagram

Cardinality of relationships

It can be used to identify relationships between entity types. The cardinality of relationships is given in Fig. 3.17.

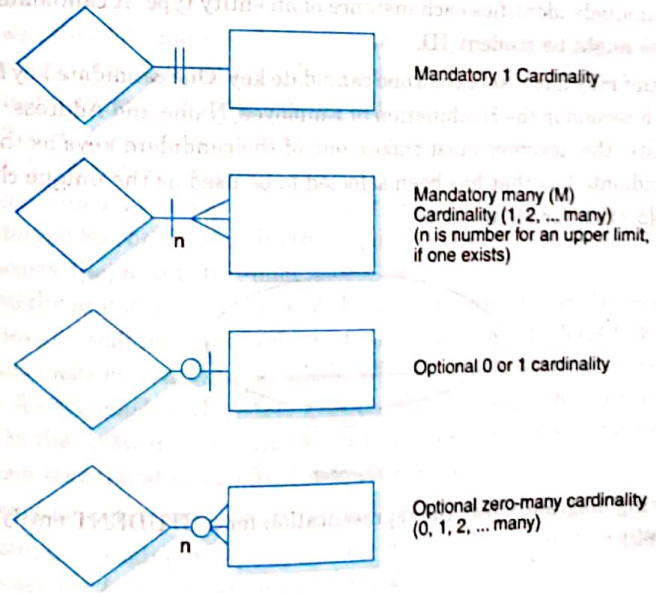


Fig. 3.17: Relationship cardinality [HOFF99]

Attributes

Each entity type has a set of attributes associated with it. An attribute is a property or characteristic of an entity that is of interest to the organization. Following are some typical entity types and associated attributes:

STUDENT: Student_ID, Student_Name, Address, Phone_Number

EMPLOYEE: Employee_ID, Employee_Name, Address.

We use an initial capital letter, followed by lowercase letters, and nouns in naming an attribute. In E-R diagram, we can visually represent an attribute by placing its name as an ellipse with a line connecting it to the associated entity. Notation for attribute is

Attribute

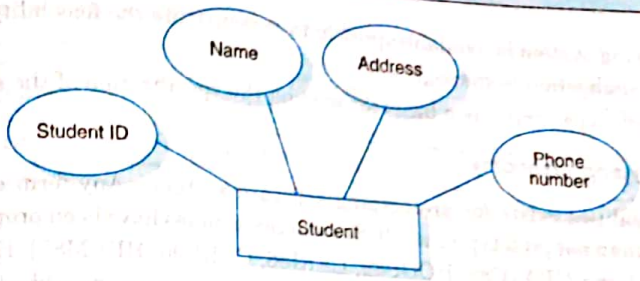
Candidate keys and Identifier

Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. A candidate key is an attribute (or combination of attributes) that uniquely identifies each instance of an entity type. A candidate key for a STUDENT entity type might be student_ID.

Some entities may have more than one candidate key. One candidate key for EMPLOYEE is Employee_ID, a second is the combination of Employee_Name and Address. If there is more than one candidate, the designer must choose one of the candidate keys as the identifier. An identifier is a candidate key that has been selected to be used as the unique characteristic of an entity type. Notation for identifier is

Identifier

The following diagram shows the representation for a STUDENT entity type using this notation [HOFF99].



Using entity relationship diagram to represent data is still an important technique today. It should not, however, be used in isolation, but together with techniques that fully represent the business objects we encounter every day.

The data flow diagram and the E-R diagram, each highlight a different aspect of the same system. As a consequence, there are one-to-one correspondences that must be checked to ensure that an E-R diagram and a data flow diagram are consistent over all applications. This suggests that it is desirable to use both methods such that we can view the logical issues from the two perspectives generated by these approaches.

3.5.4 Software Prototyping

Prototyping is the technique of constructing a partial implementation of a system so that customers, users, or developers can learn more about a problem or a solution to that problem. It is a partial implementation because if it were full implementation, it would be the system, not a prototype of it.

It allows users to explore and criticize proposed systems before undergoing the cost of a full-scale development. The field of prototyping software systems has emerged around two prototyping technologies, *i.e.*, throw-away and evolutionary. In throw-away approach, the prototype software is constructed in order to learn about the problem or its solution and is usually discarded after the desired knowledge is gained. In the evolutionary approach, the prototype is constructed in order to learn about the problem or its solution in successive steps. Once the prototype has been used and the requisite knowledge is gained, the prototype is then adapted to satisfy the, now better-understood, needs. The prototype is then used again, more is learned, and the prototype is re-adapted. This process repeats indefinitely until the prototype system satisfies all needs and thus evolves into the real system [DAV190]. Hence, in evolutionary prototyping the focus is on achieving functionality for demonstrating a portion of the system to the end user for feedback and system growth. The prototype emerges as the actual system downstream in the software life cycle. As with each iteration in development, functionality is added and then translated to an efficient implementation.

The benefits of developing a prototype early in the software process are [SOMM96]

1. Misunderstanding between software developers and customers may be identified as the system functions are demonstrated.
2. Missing user requirements may be detected.
3. Difficult-to-use or confusing user requirements may be identified and refined.

4. A working system is available quickly to demonstrate the feasibility and usefulness of the application to management.
5. The prototype serves as a basis for writing the specification of the system.

Software prototyping taxonomy

A range of possibilities exists for prototyping software systems. Any form of prototyping is perceived better than not prototyping at all. Several taxonomies have been proposed and served as basis for prototyping [RATC88, HOOP89, CERI86, CARE90, HEKM87]. However, the two most popular prototyping approaches mentioned earlier are briefly described as:

1. Throw-away prototyping: In this approach, prototype is constructed with the idea that it will be discarded, after the analysis is complete, and the final system is built from the scratch. This prototype is generally built quickly so as to enable the user to rapidly interact with the requirements determination early and thoroughly. Since the prototype will ultimately be discarded, it need not necessarily be fast operating, maintainable and having extensive fault tolerant capabilities.

During the requirement phase, a quick and dirty throw-away prototype can be constructed and given to user in order to determine the feasibility of a requirement, validate that a particular function is really necessary, uncover missing requirements and determine the viability of a user interface. During preliminary and detailed design, a quick and dirty prototype can be built to give a feeling and overview of final system to the user. Here, development of prototype should be quick, because its advantage exists only if results from its use are available in a timely fashion. It can be dirty because there is no justification for building quality into a product that will be discarded. Among the dirty characteristics to be considered are no design, no comments, no test plans, no idea about coupling and cohesion etc.

The most common steps for this approach are: (*i*) Writing a preliminary SRS (*ii*) implementing the prototype based on those requirements (*iii*) achieving user experience with the prototype (*iv*) Writing the real SRS and then (*v*) developing the real product.

2. Evolutionary prototyping: In this approach, the prototype is built with the idea that it will eventually be converted into the final system. It will not be built in a "dirty" fashion. The evolutionary prototype evolves into the final product, and thus it must exhibit all the quality attributes of the final product and must follow the traditional life cycle. It is required to deploy the product, obtain experience using it, then based on that experience go back and redo the requirements, redesign, recode, retest, and redeploy. After gaining more experience it is time to repeat the entire process again. This ensures the creation of all necessary documents and the presence of all necessary reviews. In fact, the only shortcuts that should be taken in building evolutionary prototypes are (*i*) building only those parts of the product that are understood (leaving other parts to later generations of the prototype) (*ii*) lowering the importance of performance. Using this will increase the probability that version $i + 1$ will meet user's real needs because users have already used version i and supplied feedback on its performance.

The differences between these two approaches are given below:

Sr. No.	Approach and characteristics	Throwaway	Evolutionary
1	Development approach	Quick and dirty , No rigor	No Sloppiness, Rigorous
2	What to build	Build only difficult parts	Build understood parts first and build on solid foundations
3	Design drivers	Optimize development time	Optimize modifiability
4	Ultimate Goal	Throw it away	Evolve it

Prototyping pitfalls

Prototyping has not been as successful as anticipated in some organisations for a variety of reasons [TOZE87]. Training, efficiency, applicability, and behaviour can each have a negative impact on using software prototyping techniques.

A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort. Prototyping can have execution inefficiencies with the associated tools and this question may be argued as a negative aspect of prototyping.

This new approach of providing feedback early to the end user may result in a problem related to the behavior of the end user and developers. An end user with a previously unfortunate system development effort can be biased in future interactions with development teams.

Prototyping opportunities

Not to prototype at all should simply not be an option in software development. The benefits of software prototyping are obvious and established. The end user cannot throw the ambiguous and incomplete software needs and expect the development team to return the finished software system after some period of time with no problems in the deliverables.

One of the major problems incorporating this technology is the large investment that exists in software system maintenance. The idea of completely re-engineering an existing software system with current technology is not feasible. There is, however, a threshold that exists where the expected life span of a software system justifies that the system would be better maintained after being re-engineered in this technology. Total re-engineering should be planned rather than as a reaction to a crisis situation. At minimum, prototyping technology could be used on critical portion of an existing software system. This minimal approach could be used as a means to transition an organisation to total re-engineering.

Software prototyping must be integrated within an organisation through training, case studies, and library development. In situations where this full range of commitment to this technology is lacking, e.g. only developers training is provided, when problems begin to arise in using the technology a normal reaction of management is to revert back to what has worked in the past.

The end user involvement becomes enhanced when changes in requirements can be prototyped and agreed to before any development proceeds. Similarly, during development of

the actual system or even later into maintenance, should the requirements change; the proto is enhanced and agreed to before the actual changes become confirmed.

3.6 REQUIREMENTS DOCUMENTATION

Requirements documentation is very important activity after the requirements elicitation a analysis. This is the way to represent requirements in a consistent format. Requirements do ment is called Software Requirements Specification (SRS).

The SRS is a specification for a particular software product, program or set of progra that performs certain functions in a specific environment. It serves a number of purpo depending on who is writing it. First, the SRS could be written by the customer of a syste Second, the SRS could be written by a developer of the system. The two scenarios create entir different situations and establish entirely different purposes for the document. First case, S is used to define the needs and expectations of the users. The second case, SRS is written different purpose and serve as a contract document between customer and developer.

This reduces the probability of the customer being disappointed with the final prod The SRS written by developer (second case) is of our interest and discussed in the subsequ sections.

3.6.1 Nature of the SRS

The basic issues that SRS writer (s) shall address are the following:

1. **Functionality:** What the software is supposed to do?
2. **External interfaces:** How does the software interact with people, the system's har ware, other hardware, and other software?
3. **Performance:** What is the speed, availability, response time, recovery time, etc various software functions?
4. **Attributes:** What are the considerations for portability, correctness, maintainabil security, reliability etc.?
5. **Design constraints imposed on an implementation:** Are there any requi standards in effect, implementation language, policies for database integrity, resou limits, operating environment (s) etc.?

Since the SRS has a specific role to play in the software development process, S writer(s) should be careful not to go beyond the bounds of that role. This means the SRS

1. should correctly define all the software requirements. A software requirement m exist because of the nature of the task to be solved or because of a special characteri of the project.
2. should not describe any design or implementation details. These should be describ in the design stage of the project.
3. should not impose additional constraints on the software. These are properly specif in other documents such as a software quality assurance plan.

Therefore, a properly written SRS limits the range of valid designs, but does not spe any particular design.

3.6.2 Characteristics of a good SRS

The SRS should be:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Each of the above mentioned characteristics is discussed below: [THAY97, IEEE87, IEEE97].

Correct

The SRS is correct if, and only if; every requirement stated therein is one that the software shall meet. There is no tool or procedure that assures correctness. If the software must respond to all button presses within 5 seconds and the SRS stated that "the software shall respond to all buttons presses within 10 second", then that requirement is incorrect.

Unambiguous

The SRS is unambiguous if, and only if; every requirement stated therein has only one interpretation. Each sentence in the SRS should have the unique interpretation. Imagine that a sentence is extracted from the SRS, given to ten people who are asked for their interpretation. If there is more than one such interpretation, then that sentence is probably ambiguous.

In cases, where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific. The SRS should be unambiguous to both those who create it and to those who use it. However, these groups often do not have the same background and therefore do *not tend to describe* software requirements in the same way.

Requirements are often written in natural language (for example, english). Natural language is inherently ambiguous. A natural language SRS should be reviewed by an independent party to identify ambiguous use of a language so that it can be corrected. This can be avoided by using a particular requirement specification language. Its language processors automatically detect many lexical, syntactic, and semantic errors. Disadvantage is the time required to learn the language which may also not be understandable to the customers/users. Moreover, these languages tend to be better at expressing only certain types of requirements and addressing certain types of systems.

Complete

The SRS is complete if, and only if; it includes the following elements:

1. All significant requirements, whether relating to functionality, performance, design constraints, attributes or external interfaces.

2. Definition of their responses of the software to all realizable classes of input data in *all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid values.*
3. Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

Consistent

The SRS is consistent if, and only if, no subset of individual requirements described in it conflict. There are three types of likely conflicts in the SRS:

1. The specified characteristics of real-world objects may conflict. For example.
   (a) The format of an output report may be described in one requirement as tabular but in another as textual.
   (b) One requirement may state that all lights shall be green while another states that all lights shall be blue.
2. There may be logical or temporal conflict between two specified actions, for example,
   (a) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
   (c) One requirement may state that "A" must always follow "B", while another requires that "A and B" occur simultaneously.
3. Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program's request for a user input may be called a "prompt" in one requirement and a "cue" in another. The use of standard terminology and definitions promotes consistency.

Ranked for importance and/or stability

The SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

Typically, all requirements are not equally important. Some requirements may be essential, especially for life critical applications, while others may be desirable. Each requirement should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of requirements as essential, conditional and optional.

Verifiable

The SRS is verifiable, if and only if, every requirement stated therein is verifiable. A requirement is verifiable, if and only if, there exists some finite cost-effective process with which a person or machine can check that the software meets the requirements. In general any ambiguous requirement is not verifiable.

Non-verifiable requirements include statement, such as "works well", "good human interface", and "shall usually happen". These requirements cannot be verified because it is impossible to define the terms "good", "well", or "usually". The statement that "the program shall never enter an infinite loop" is non-verifiable because the testing of this quality is theoretically impossible.

An example of a verifiable statement is "output of the program shall be produced within 20 seconds of event × 60% of the time; and shall be produced within 30 seconds of event × 100% of the time." This statement can be verified because it uses concrete terms and measurable quantities.

If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

Modifiable

The SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.

The requirements should not be redundant. Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make an SRS more readable, but a problem can arise when the redundant document is updated. For instance, a requirement may be altered in only one of the places out of the many places where it appears.

The SRS then becomes inconsistent. Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.

Traceable

The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. Two types of traceability are recommended.

1. Backward traceability: This depends upon each requirement explicitly referencing its source in earlier documents.
2. Forward traceability: This depends upon each requirement in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

3.6.3 Organization of the SRS

The Institute of Electrical and Electronics Engineers (IEEE) has published guidelines and standards to organize an SRS document [IEEE87, IEEE94]. Different projects may require their requirements to be organized differently, that is, there is no one method that is suitable for all projects. It provides different ways of structuring the SRS. The first two sections of the SRS are the same in all of them. The specific tailoring occurs in section 3 entitled "specific requirements". The general organisation of an SRS is given in Fig. 3.18 [IEEE93].

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms, and Abbreviations
   - 1.4 References
   - 1.5 Overview
2. The Overall Description
   - 2.1 Product Perspective
     - 2.1.1 System Interfaces
     - 2.1.2 Interfaces
     - 2.1.3 Hardware Interfaces
     - 2.1.4 Software Interfaces
     - 2.1.5 Communications Interfaces
     - 2.1.6 Memory Constraints
     - 2.1.7 Operations
     - 2.1.8 Site Adaptation Requirements
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and Dependencies.
   - 2.6 Apportioning of Requirements
3. Specific Requirements
   - 3.1 External interfaces
   - 3.2 Functions
   - 3.3 Performance Requirements
   - 3.4 Logical Database Requirements
   - 3.5 Design Constraints
     - 3.5.1 Standards Compliance
   - 3.6 Software System Attributes
     - 3.6.1 Reliability
     - 3.6.2 Availability
     - 3.6.3 Security
     - 3.6.4 Maintainability
     - 3.6.5 Portability
   - 3.7 Organising the Specific Requirements
     - 3.7.1 System Mode
     - 3.7.2 User Class
     - 3.7.3 Objects
     - 3.7.4 Feature
     - 3.7.5 Stimulus
     - 3.7.6 Response
     - 3.7.7 Functional Hierarchy
   - 3.8 Additional Comments
4. Change Management Process
5. Document Approvals
6. Supporting Information

Fig. 3.18: Organisation of SRS [IEEE-std. 830-1993]

1. Introduction

The following subsections of the Software Requirements Specifications (SRS) document should provide an overview of the entire SRS.

1.1 Purpose

Identify the purpose of this SRS and its intended audience. In this subsection, describe the purpose of the particular SRS and specify the intended audience for the SRS.

1.2 Scope

In this subsection:

(*i*) Identify the software product(s) to be produced by name

(*ii*) Explain what the software product(s) will, and, if necessary, will not do

(*iii*) Describe the application of the software being specified, including relevant benefits, objectives, and goals

(*iv*) Be consistent with similar statements in higher-level specifications if they exist.

1.3 Definitions, Acronyms, and Abbreviations

Provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendices in the SRS or by reference to documents. This information may be provided by reference to an Appendix.

1.4 References

In this subsection:

(*i*) Provide a complete list of all documents referenced elsewhere in the SRS

(*ii*) Identify each document by title, report number (if applicable), date, and publishing organisation

(*iii*) Specify the sources from which the references can be obtained.

This information can be provided by reference to an appendix or to another document.

1.5 Overview

In this subsection:

(*i*) Describe what the rest of the SRS contains

(*ii*) Explain how the SRS is organised.

2. The Overall Description

Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in section 3, and makes them easier to understand.

2.1 Product Perspective

Put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection relates the require-

ments of the larger system to functionality of the software and identifies interfaces between that system and the software.

A block diagram showing the major components of the large system, interconnections, and external interfaces can be helpful.

The following subsections describe how the software operates inside various constraints.

2.1.1 System Interfaces

List each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

2.1.2 Interfaces

Specify:

(*i*) The logical characteristics of each interface between the software product and its users.

(*ii*) All the aspects of optimizing the interface with the person who must use the system.

2.1.3 Hardware Interfaces

Specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics. It also covers such matters as what devices are to be supported, how they are to be supported and protocols.

2.1.4 Software Interfaces

Specify the use of other required software products and interfaces with other application systems. For each required software product, include:

(*i*) Name

(*ii*) Mnemonic

(*iii*) Specification number

(*iv*) Version number

(*v*) Source

For each interface, provide:

(*i*) Discussion of the purpose of the interfacing software as related to this software product

(*ii*) Definition of the interface in terms of message content and format.

2.1.5 Communications Interfaces

Specify the various interfaces to communications such as local network protocols, etc.

2.1.6 Memory Constraints

Specify any applicable characteristics and limits on primary and secondary memory.

2.1.7 Operations

Specify the normal and special operations required by the user such as:

(*i*) The various modes of operations in the user organisation

(*ii*) Periods of interactive operations and periods of unattended operations

(*iii*) Data processing support functions

(*iv*) Backup and recovery operations.

2.1.8 Site Adaptation Requirements

In this section:

(*i*) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode

(*ii*) Specify the site or mission-related features that should be modified to adapt the software to a particular installation.

2.2 Product Functions

Provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product.

For clarity:

(*i*) The functions should be organised in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.

(*ii*) Textual or graphic methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product but simply shows the logical relationships among variables.

2.3 User Characteristics

Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Do not state specific requirements but rather provide the reasons why certain specific requirements are later specified in sections 3.

2.4 Constraints

Provide a general description of any other items that will limit the developer's options. These can include:

(*i*) Regulatory policies

(*ii*) Hardware limitations (for example, signal timing requirements)

(*iii*) Interface to other applications

(*iv*) Parallel operation

(*v*) Audit functions

(*vi*) Control functions

(*vii*) Higher-order language requirements

(*viii*) Signal handshake protocols (for example, XON-XOFF, ACK-NACK)

(*ix*) Reliability requirements

(*x*) Criticality of the application

(*xi*) Safety and security considerations.

2.5 Assumptions and Dependencies

List each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system would be available on the hardware designated for the software product. If, in fact, the operating system were not available, the SRS would then have to change accordingly.

2.6 Apportioning of Requirements

Identify requirements that may be delayed until future versions of the system.

3. Specific Requirements

This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or both external systems. These requirements should include at a minimum a description of every input into the system, every output from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:

(*i*) Specific requirements should be stated with all the characteristics of a good SRS

- correct
- unambiguous
- complete
- consistent
- ranked for importance and/or stability
- verifiable
- modifiable
- traceable

(*ii*) Specific requirements should be cross-referenced to earlier documents that relate

(*iii*) All requirements should be uniquely identifiable

(*iv*) Careful attention should be given to organizing the requirements to maximize readability.

Before examining specific ways of organising the requirements it is helpful to understand the various items that comprise requirements as described in the following sebsections.

3.1 External Interfaces

This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in *section 2 but does* not repeat information there. It contains both content and format as follows:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy and/or tolerance
- Units of measure

- Timing
- Relationships to other inputs/outputs
- Screen formats/organisation
- Window formats/organisation
- Data formats
- Command formats
- End messages.

3.2 Functions

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall... These include:

- Validity checks on the inputs
- Exact sequence of operations
- Responses to abnormal situation, including
  - Overflow
  - Communication facilities
  - Error handling and recovery
- Effect of parameters
- Relationship of outputs to inputs, including
  - Input/Output sequences
  - Formulas for input to output conversion.

It may be appropriate to partition the functional requirements into sub-functions or sub-processes. This does not imply that the software design will also be partitioned that way.

3.3 Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:

(*i*) The number of terminals to be supported

(*ii*) The number of simultaneous users to be supported

(*iii*) Amount and type of information to be handled

Static numerical requirements are sometimes identified under a separate section entitled capacity.

Dynamic numerical requirements may include, for example, the number of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms:

For example,

95% of the transactions shall be processed in less than 1 second rather than,

An operator shall not have to wait for the transaction to complete.

(Note: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function).

3.4 Logical Database Requirements

This section specifies the logical requirements for any information that is to be placed into a database. This may include:

- Types of information used by various functions
- Frequency of use
- Accessing capabilities
- Data entities and their relationships
- Integrity constraints
- Data retention requirements.

3.5 Design Constraints

Specify design constraints that can be imposed by other standards, hardware limitations, etc.

3.5.1 Standards Compliance

Specify the requirements derived from existing standards or regulations. They might include:

(*i*) Report format
(*ii*) Data naming
(*iii*) Accounting procedures
(*iv*) Audit Tracing

For example, this could specify the requirement for software to processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standard. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.6 Software System Attributes

There are a number of quality attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified. Fig. 3.19 has the definitions of the quality attributes of the software discussed in this subsection [ROBE02]. The following items provide a partial list of examples.

3.6.1 Reliability

Specify the factors required to establish the required reliability of the software system at time of delivery.

3.6.2 Availability

Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

3.6.3 Security

Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:

- Utilize certain cryptographic techniques
- Keep specific log or history data sets
- Assign certain functions to different modules
- Restrict communications between some areas of the program
- Check data integrity for critical variables.

3.6.4 Maintainability

Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

3.6.5 Portability

Specify attributes of software that relate to the ease of parting the software to other host machines and/or operating systems. This may include:

- Percentage of components with host-dependent code
- Percentage of code that is host dependent
- Use of a proven portable language
- Use of a particular compiler or language subset
- Use of a particular operating system.

S. No.	Quality Attributes	Definition
1.	Correctness	extent to which program satisfies specifications, fulfills user's mission objectives
2.	Efficiency	amount of computing resources and code required to perform function
3.	Flexibility	effort needed to modify operational program
4.	Interoperability	effort needed to couple one system with another
5.	Reliability	extent to which program performs with required precision
6.	Reusability	extent to which it can be reused in another application
7.	Testability	effort needed to test to ensure performance as intended
8.	Usability	effort required to learn, operate, prepare input, and interpret output
9.	Maintainability	effort required to locate and fix an error during operation
10.	Portability	effort needed to transfer from one hardware or software environment to another.
11.	Integrity/security	extent to which access to software or data by unauthorised people can be controlled.

Fig. 3.19: Definitions of quality attributes

3.7 Organising the Specific Requirements

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organising these in a manner optimal for understanding. There is no one optimal organisation for all systems. Different classes of systems lend themselves to different organisations of requirements. Some of these organisations are described in the following subclasses.

3.7.1 System mode

Some systems behave quite differently depending on the mode of operation. When organising by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode.

3.7.2 User class

Some systems provide different sets of functions to different classes of users.

3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. Associated with each object is a set of attributes and functions. These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.

3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. Each feature is generally described as sequence of stimulus-response pairs.

3.7.5 Stimulus

Some systems can be best organised by describing their functions in terms of stimuli.

3.7.6 Response

Some systems can be best organised by describing their functions in support of the generation of a response.

3.7.7 Functional hierarchy

When none of the above organisational schemes prove helpful, the overall functionality can be organised into a hierarchy of functions organised by either common inputs, common outputs or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data.

3.8 Additional Comments

Whenever a new SRS is contemplated, more than one of the organisational techniques given in 3.7 may be appropriate. In such cases, organise the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organisation. For example, when organising by mode, finite state machines or state charts may prove helpful, when organising by object, object-oriented analysis may prove helpful; when organising by feature, stimulus-response sequences may prove helpful; when organising by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines below, those sections called "Functional Requirement i" may be described in native language, in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, Outputs.

4. Change Management Process

Identify the change management process to be used to identify, log, evaluate, and update the SRS to reflect changes in project scope and requirements.

5. Document Approval

Identify the approvers of the SRS document. Approver's name, signature, and date should be used.

6. Supporting Information

The supporting information makes the SRS easier to use. It includes:

- Table of Contents
- Index
- Appendices

The appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:

(*a*) Sample I/O formats, descriptions of cost analysis studies, results of user surveys

(*b*) Supporting or background information that can help the readers of the SRS

(*c*) A description of the problems to be solved by the software

(*d*) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendices are included, the SRS should explicitly state whether or not the appendices are to be considered part of the requirements.

Tables on the following pages provide alternate ways to structure section 3 on the specific requirements.

Outline for SRS Section 3 Organised by Mode: Version 1

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Mode 1
    - 3.2.1.1 Functional requirements 1.1 .....
    - 3.2.1.*n* Functional requirements 1.*n*
  - 3.2.2 Mode 2 .....
  - 3.2.*m* Mode *m*
    - 3.2.*m*.1 Functional requirement *m*.1 .....
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Mode: Version 2

3 Specific Requirements
- 3.1 Functional Requirements
  - 3.1.1 Mode 1
    - 3.1.1.1 External interfaces
    - 3.1.1.1 User Interfaces
    - 3.1.1.2 Hardware interfaces
    - 3.1.1.3 Software interfaces
    - 3.1.1.4 Communications interfaces
    - 3.1.1.2 Functional Requirement
      - 3.1.1.2.1 Functional requirement 1 .....
      - 3.1.1.2.1.*n* Functional requirement *n*
    - 3.1.1.3 Performance
  - 3.1.2 Mode 2 .....
  - 3.1.*m* Mode *m*
- 3.2 Design constraints
- 3.3 Software system attributes
- 3.4 Other requirements.

Outline for SRS Section 3 Organised by User Class

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 User class 1
    - 3.2.1.1 Functional requirements 1.1 .....
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 User class 2 .....
  - 3.2.*m* User class *m*
    - 3.2.*m*.1 Functional requirement *m*.1 .....
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Object

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Classes/Objects
  - 3.2.1 Class/Object 1
    - 3.2.1.1 Attributes (direct or inherited)
      - 3.2.1.1.1 Attribute 1 .....
      - 3.2.1.1.*n* Attribute *n*
    - 3.2.1.2 Functions (services, methods, direct or inherited)
      - 3.2.1.2.1 Functional requirement 1.1 .....
      - 3.2.1.2.*m* Functional requirement 1.*m*
    - 3.2.1.3 Messages (communications received or sent)
  - 3.2.2 Class/Object 2 .....
  - 3.2.*p* Class/Object *p*
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Feature

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 System Features
  - 3.2.1 System Feature 1
    - 3.2.1.1 Introduction/Purpose of feature
    - 3.2.1.2 Stimulus/Response sequence
    - 3.2.1.3 Associated functional requirements
      - 3.2.1.3.1 Functional requirement 1 .....
      - 3.2.1.3.*n* Functional requirement *n*
  - 3.2.2 System Feature 2 .....
  - 3.2.*m* System Feature *m* .....
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Stimulus

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Stimulus 1
    - 3.2.1.1 Functional requirement 1.1 .....
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Stimulus 2 .....
  - 3.2.*m* Stimulus *m*
    - 3.2.*m*.1 Functional requirement *m*.1 .....
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Response

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Response 1
    - 3.2.1.1 Functional requirement 1.1 .....
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Response 2 .....
  - 3.2.*m* Response *m*
    - 3.2.*m*.1 Functional requirement *m*.1.....
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Outline for SRS Section 3 Organised by Functional Hierarchy

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Information flows
    - 3.2.1.1 Data flow diagram 1
      - 3.2.1.1.1 Data entities
      - 3.2.1.1.2 Pertinent processes
      - 3.2.1.1.3 Topology
    - 3.2.1.2 Data flow diagram 2
      - 3.2.1.2.1 Data entities
      - 3.2.1.2.2 Pertinent processes
      - 3.2.1.2.3 Topology .....
    - 3.2.1.*n* Data flow diagram *n*
      - 3.2.1.*n*.1 Data entities
      - 3.2.1.*n*.2 Pertinent processes
      - 3.2.1.*n*.3 Topology

(Contd)...

3.2.2 Process descriptions
- 3.2.2.1 Process 1
  - 3.2.2.1.1 Input data entities
  - 3.2.2.1.2 Algorithm or formula of process
  - 3.2.2.1.3 Affected data entities
- 3.2.2.2 Process 2
  - 3.2.2.2.1 Input data entities
  - 3.2.2.2.2 Algorithm or formula of process
  - 3.2.2.2.3 Affected data entities .....
- 3.2.2.*m* Process *m*
  - 3.2.2.*m*.1 Input data entities
  - 3.2.2.*m*.2 Algorithm or formula of process
  - 3.2.2.*m*.3 Affected data entities

3.2.3 Data construct specifications
- 3.2.3.1 Construct 1
  - 3.2.3.1.1 Record type
  - 3.2.3.1.2 Constituent fields
- 3.2.3.2 Construct 2
  - 3.2.3.2.1 Record type
  - 3.2.3.2.2 Constituent fields .....
- 3.2.3.*p* Construct *p*
  - 3.2.3.*p*.1 Record type
  - 3.2.3.*p*.2 Constituent fields

3.2.4 Data dictionary
- 3.2.4.1 Data element 1
  - 3.2.4.1.1 Name
  - 3.2.4.1.2 Representation
  - 3.2.4.1.3 Units/Format
  - 3.2.4.1.4 Precision/Accuracy
  - 3.2.4.1.5 Range
- 3.2.4.2 Data element 2
  - 3.2.4.2.1 Name
  - 3.2.4.2.2 Representation
  - 3.2.4.2.3 Units/Format
  - 3.2.4.2.4 Precision/Accuracy
  - 3.2.4.2.5 Range .....
- 3.2.4.*q* Data element *q*
  - 3.2.4.*q*.1 Name
  - 3.2.4.*q*.2 Representation
  - 3.2.4.*q*.3 Units/Format
  - 3.2.4.*q*.4 Precision/Accuracy
  - 3.2.4.*q*.5 Range

3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3 Showing Multiple Organisations

3 Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 User class 1
    - 3.2.1.1 Feature 1.1
      - 3.2.1.1.1 Introduction/Purpose of feature
      - 3.2.1.1.2 Stimulus/Response sequence
      - 3.2.1.1.3 Associated functional requirements
    - 3.2.1.2 Feature 1.2
      - 3.2.1.2.1 Introduction/Purpose of feature
      - 3.2.1.2.2 Stimulus/Response sequence
      - 3.2.1.2.3 Associated functional requirements .....
    - 3.2.1.*m* Feature 1.*m*
      - 3.2.1.*m*.1 Introduction/Purpose of feature
      - 3.2.1.*m*.2 Stimulus/Response sequence
      - 3.2.1.*m*.3 Associated functional requirements
  - 3.2.2 User class 2 .....
  - 3.2.*n* User class *n* .....
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Software system attributes
- 3.6 Other requirements

3.7 REQUIREMENTS VALIDATION

After the completion of SRS document, we may like to check the document for

- completeness and consistency
- conformance to standards
- requirements conflicts
- technical errors
- ambiguous requirements

The objective of requirements validation is to certify that the SRS document is an acceptable document of the system to be implemented. This helps us to find errors in the document and improves the quality of the software development process. Sometimes we confuse between analysis and validation. However, analysis works with raw requirements as elicited from the various stakeholders whereas validation works with a final draft of the SRS document

with negotiated and agreed requirements. The inputs and outputs of validation process are shown in figure 3.20. There are many requirements validation techniques and few are discussed in subsequent subsections.

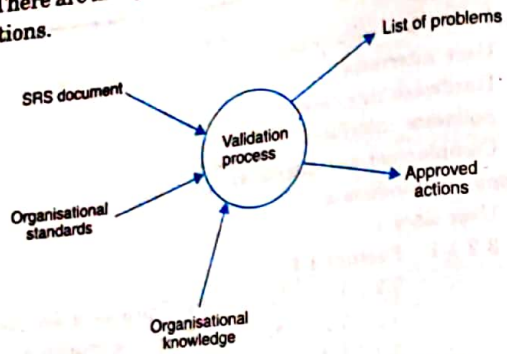


Fig. 3.20: Validation process with inputs and outputs

SRS document: It should be a final draft, not an unfinished draft and should be formatted/organised as per IEEE standards.

Organisational standards: Every organisation should have some quality standards for SRS document and other activities. These standards will be used for reviewing during validation.

Organisational knowledge: Knowledge, often implicit, of the organisation which may be used to judge the realism of the requirements.

Problem list: List of discovered problems in the requirements document.

Approved actions: List of approved actions in response to the requirements problems. Some problems may have several corrective actions; some problems may have no associated actions.

3.7.1 Requirements Reviews

This is a popular requirements validation technique where a group of people will read the SRS document and look for possible problems. These identified problems may be discussed in the group and some actions may also be approved in order to get rid of these problems. The requirements review process is given in Figure 3.21.

(*i*) **Plan review:** The review team is selected and time and place for review meeting fixed.

(*ii*) **Distribute SRS document:** The SRS document is distributed to all the members.

(*iii*) **Read SRS document:** Each member should read the document carefully to find conflicts, omissions, inconsistencies, deviations from standards and other problems.

(*iv*) **Organise review meeting:** Each member presents his/her views and identifies problems. The problems are discussed and a set of actions to address the problem is approved.

(*v*) **Follow-up actions:** The chairperson of the team checks that the approved actions have been carried out.

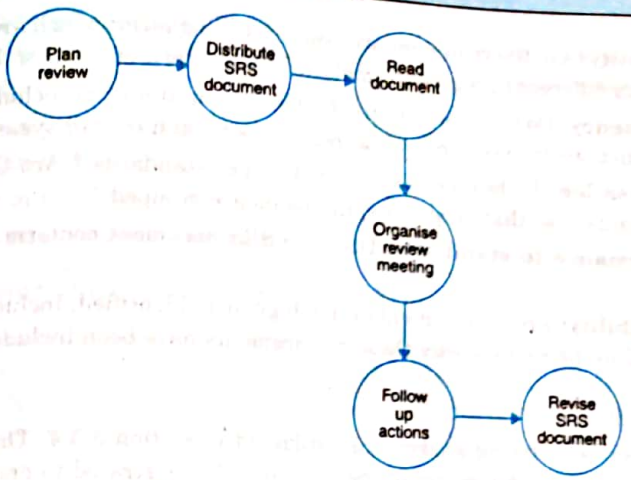


Fig. 3.21: Requirements review process

(*vi*) **Revise SRS document:** The SRS document is revised to reflect the approved actions. At this stage, it may be accepted or may be reviewed.

Problem actions: Some of the problems actions are discussed below :

(*i*) **Requirements clarification:** The requirement may be badly expressed or may have accidentally omitted during the process of elicitation.

(*ii*) **Missing information:** Some information is missing from the SRS document. It is the responsibility of the requirements engineers who are revising the document to find this information from system stakeholders.

(*iii*) **Requirements conflict:** There is a conflict between requirements. The stakeholders involved must negotiate to resolve this conflict.

(*iv*) **Unrealistic requirement:** The requirement does not appear to be implementable with the technology available or within other constraints on the system. Stakeholders must be consulted to decide how to make the requirement more realistic.

(*v*) **Security issues:** Review team may comment on security issues. The security and safety of the system is essential for the survival of the system. The reviewers may review the system in accordance with some security standards.

Membership of review team: The team should involve a number of stakeholders drawn from different backgrounds. People from different backgrounds bring different skills and knowledge to review the SRS document. Stakeholders feel involved in the process and develop an understanding of the needs of other stakeholders. Review team should always involve at least one domain expert and an enduser.

Review checklists

(*i*) **Understandability:** Can readers of the document understand what the requirements mean ?

(*ii*) **Redundancy:** Is information unnecessarily repeated in the SRS document ?

(*iii*) **Completeness:** Does the checker know of any missing requirements or is there any information missing from individual requirement descriptions ?

(*iv*) **Ambiguity:** Are the requirements expressed using terms which are clearly defined? Could readers from different backgrounds make different interpretations of the requirements?

(*v*) **Consistency:** Do the descriptions of different requirements include contradictions? Are there contradictions between individual requirements and overall system requirements?

(*vi*) **Organisation:** Is document structured as per standards ? Are the descriptions of requirements organised so that related requirements are grouped ?

(*vii*) **Conformance to standards:** Does the SRS document conform to defined standards ?

(*viii*) **Traceability:** Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included ?

3.7.2 Prototyping

Prototyping is also used during analysis as explained in section 5.3.4. This is an expensive exercise because an executable model of the system is demonstrated to endusers, customers and other review team members. Validation prototype should be reasonably complete and efficient and should be used as the required system. Some documents and training may also be necessary to get good results. (For details please refer to section 5.3.4). Prototyping is also a popular validation technique if a prototype has been developed during the requirements elicitation and analysis stage.

3.8 REQUIREMENTS MANAGEMENT

As we have often discussed changes in requirements are inevitable. Many times, we are forced to make changes due to factors beyond our control. Every change is painful and requires some systematic way to handle it. Moreover, after the installation of system, requests for new requirements or changed requirements start and may create challenging situations for the developers. It is difficult to predict what effects the new system will have on the organisation. After the experience of the installed system, customer may find new needs and priorities.

Requirements management is a new term which has been rapidly adopted by industry. It is the process of understanding and controlling changes to system requirements. Most of times, requirements are not independent ; but are dependent on each other. Hence, one change may have implications on other dependent requirements and thus make the task much more difficult and challenging.

3.8.1 Enduring and Volatile Requirements

Requirements are broadly classified in two categories *i.e.*, enduring and volatile.

(*i*) **Enduring requirements:** These are core requirements and are related to main activity of the organisation. For a library management system, issue/return a book, cataloging etc. are core activities and are stable for any system. Hence, enduring requirements are stable and are not changed easily.

(*ii*) **Volatile requirements.** These requirements are likely to change during software development life cycle or even after delivery of the product. There are many reasons for such changes some of the reasons are :

- Changes to the environment
- Changes in technology
- Changes in policies
- Changes in customer's expectations

3.8.2 Requirements Management Planning

Requirements management planning is very critical, but important for the success of any project. The process of requirement management ends when the final product is released, and the customer is fully satisfied. However, the fewer modifications in requirements, may also be better for everybody. We should be able to trace our requirements all the time. When a customer comes up with an additional issue, it may be too late to change a requirement or add a new one—the workload and costs are simply too large to make it worth it. This remains subject of negotiation between us and the customer; but our task is to know exactly what would be the effect of implementing new requirements, and to translate in to the language of the customer. The customer may not be receptive when he sees how many code lines need to be changed, but he may understand when we tell him how much this will cost.

Tracing requirements also involves additional tests, performed from time to time, to ensure that the process runs smoothly and errors are identified and corrected early on. When faced with a big project, we may have different sets of requirements, some that apply to the entire project, and some for parts of it. When a certain design is implemented for certain requirements, make a note about the effects and the alternatives it may be useful for future projects or even for the same project, if the customer is not satisfied with the result.

3.8.3 Requirements Change Management

The customers expect modern software systems to be evolvable and sufficiently flexible to accomodate changing users needs. This expectation, however, is untempered by the fact that changing requirements are recognised as a major cause of project failure. This may also have a degrading effect on the underlying design of a system. The fact of life is that software requirements change and evolve from inception to deployment.

Hence changes to requirements are carefully traced, analyzed and their effects on the overall system is properly assessed. The requirements change process should include the following activities to be carried out when a change is needed in the requirements .

- Allocating adequate resources (Assignment of responsibilities).
- Analysis of requirement changes (Menagement of changes)
- Documenting requirements (Documentation)
- Creating requirements traces throughout the project life cycle from inception to the final work products (requirements traceability).
- Establishing team communication (communication of change)
- Establishing a baseline for requirements specification (Establishment of baseline).

(*i*) **Assignment of responsibilities:** All changes should be approved or rejected by the competent authority designated for the project. It may be project manager, project lead or any other equivalent person. After the decision, work is further given to individuals to carry out desired modifications.

(*ii*) **Management of changes:** It is an important activity for the implementation of any change. Whenever a request is received for any addition/modification, a change request is created through a specified request form. The request is analyzed due to its impact on overall project cost, resources allocated and delivery schedule of the project. After the implementation of any change, a formal notice is issued for the information to all stakeholders.

(*iii*) **Documentation:** It is required to keep track of every activity of the project. Success of any software project lies in its documentation. It is very pertinent to note that without documentation, project future is in dark and with every passing day, it is leading towards disaster. Documenting requirements is an iterative process and should be carried out with utmost sincerity.

(*iv*) **Requirements traceability:** It is a technique to provide relationships between requirements, design and implementation in order to manage the effect of change and its impact on the success of the project. Every requirement should be traceable and every piece of design work may be traced to one or more requirements in the project. If we are not able to trace any implementation activity to its requirements, it indicates that either there are potential issues with the accuracy and goodness of the requirements or else the design team is implementing something that is outside the scope of the system.

(*v*) **Communication of change:** The most problematic changes to the requirements specification have been the ones that are not communicated to every stakeholder. Communication failures typically occur when we may drop a feature or change a performance requirement without communication to others. Establishment of automated e-mail notification system may inform everyone in an organised and timely manner. This may only increase the possibility of the success of any change.

(*vi*) **Establishment of baseline:** After the approval of a change, it is communicated to all. If every stakeholder agrees to the change, it is implemented and tested. Baseline is the tested version of a set of requirements representing a conceptual milestone, serves as the basis for further development. A particular version becomes baseline, when a responsible group decides to designate it as such. Baselining is simply labeling a set of requirements at specific versions and freezing them before proceeding to the next phase of development. The steps in baselines establishment are:

- Approval of changes from competent authority.
- Communicate the approval to all stakeholders.
- History of changes is maintained.
- All documents are updated to reflect the changes.

3.9 STUDENT RESULT MANAGEMENT SYSTEM—EXAMPLE

A university has decided to engage a software company for the automation of student result management system of its M. Tech. Programme. The following documents are required to be prepared.

(*i*) Problem statement
(*ii*) Context diagram
(*iii*) Data flow diagrams
(*iv*) ER diagrams
(*v*) Use case diagram
(*vi*) Use cases
(*vii*) SRS as per IEEE std. 830-1993

These seven documents may provide holistic view of the systme to be developed. The SRS will act as contract document between developers (software company) and client (University).

3.9.1 Problem Statement

The problem statement is the first document which is normally prepared by the client. It only, gives superficial view of the system as per client's perspective and expectations. It is the input to the requirement engineering process where final product is the SRS.

The problem statement of student result management system of M. Tech. (Information Technology) Programme of a University is given below:

"A University conducts a 4-semester M. Tech. (IT) program. The students are offered four theory papers and two Lab papers (practicals) during Ist, IInd and IIIrd semesters. The theory papers offered in these semesters are categorized as either 'Core' or 'Elective'. Core papers do not have an alternative subject, whereas elective papers have two other alternative subjects. Thus, a student can study any subject out of the 3 choices available for an elective paper.

In Ist, IInd and IIIrd semesters, 2 core papers and 2 elective papers are offered to each student. The students are also required to submit a term paper/minor project in IInd and IIIrd semesters each. In IVth semester the students have to give a seminar and submit a dissertation on a topic/subject area of their interest.

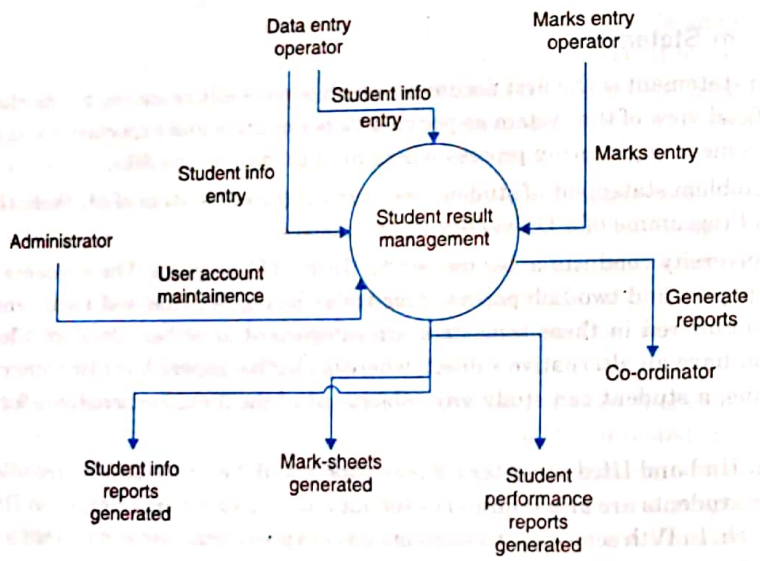
The evaluation of each subject is done out of 100 marks. During the semester, minor exams are conducted for each semester. Students are also required to submit assignments as directed by the corresponding faculty and maintain Lab records for practicals. Based on the students' performance in minor exams, assignments, Lab records and their attendance, marks out of 40 are given in each subject and practical paper. These marks out of 40 account for internal evaluation of the students. At the end of each semester major exams are conducted in each subject (theory as well as practical). These exams are evaluated out of 60 marks and account for external evaluation of the students. Thus, the total marks of a student in a subject are obtained by adding the marks obtained in internal and external evaluation.

Every subject has some credit points assigned to it. If the total marks of a student are > = 50 in a subject, he/she is considered 'Pass' in that subject otherwise the student is considered 'Fail' in that subject. If a student passes in a subject, he/she earns all the credit points assigned to that subject, but if the students fails in a subject he/she does not earn any credit point in that subject. At any time, the latest information about subjects being offered in various semesters and their credit points can be obtained from university's website.

It is required to develop a system that will manage information about subjects offered in various semesters, students enrolled in various semesters, elective (s) opted by various students in different semesters, marks and credit points obtained by students in different semesters. The system should also have the ability to generate printable mark-sheets for each student. Semester-wise detailed mark lists and student performance reports also need to be generated.

3.9.2 Context Diagram

The context diagram is given below:



The following persons are interacting with the "student result management system"

(*i*) Administrator

(*ii*) Marks entry operator

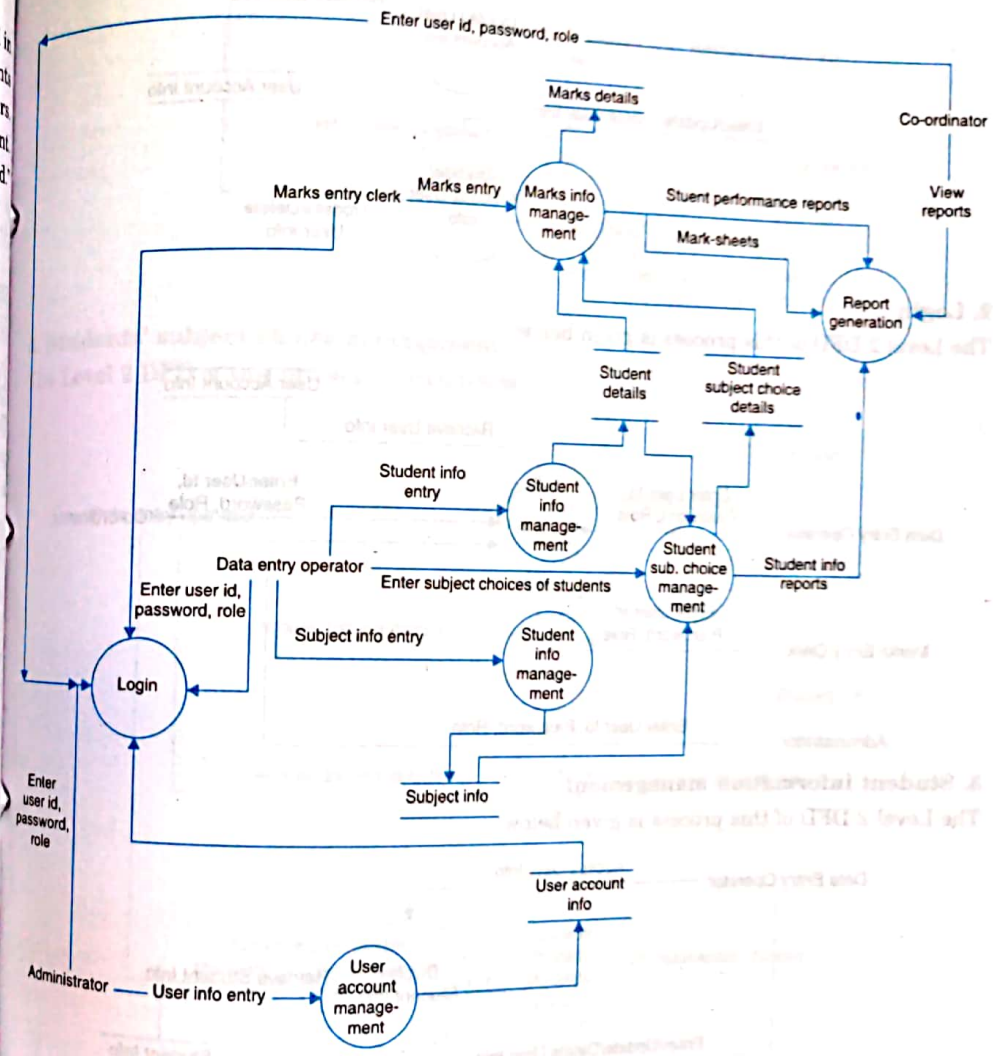
(*iii*) Data entry operator

(*iv*) Co-ordinator

3.9.3 Level-n DFD

Level-1 DFD

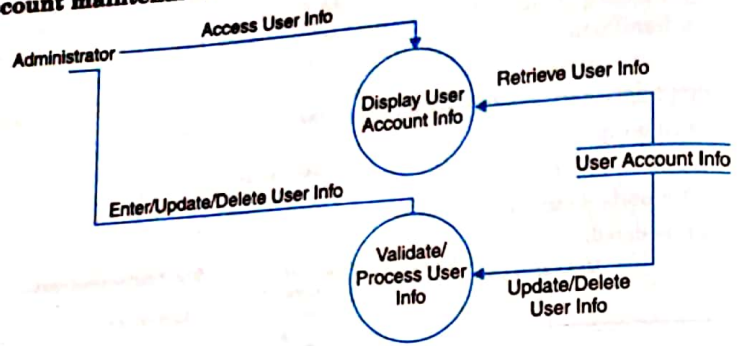
The Level-1 DFD is given below:



Level 1 DFD of result management system

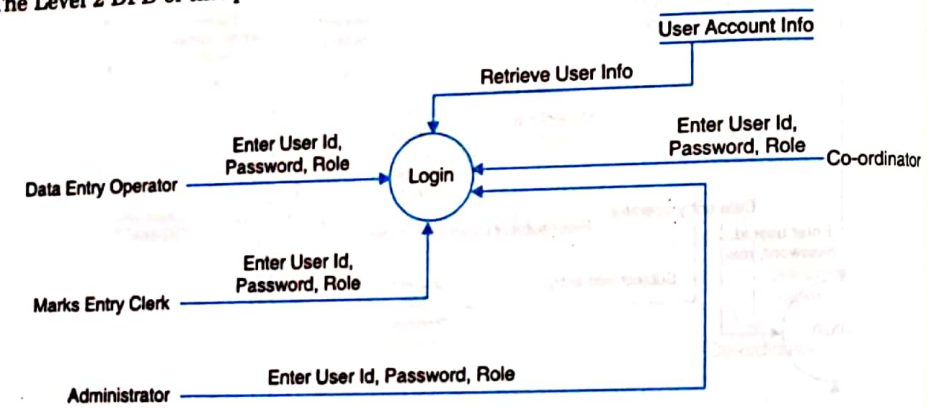
Level-2 DFDs

1. User account maintenance



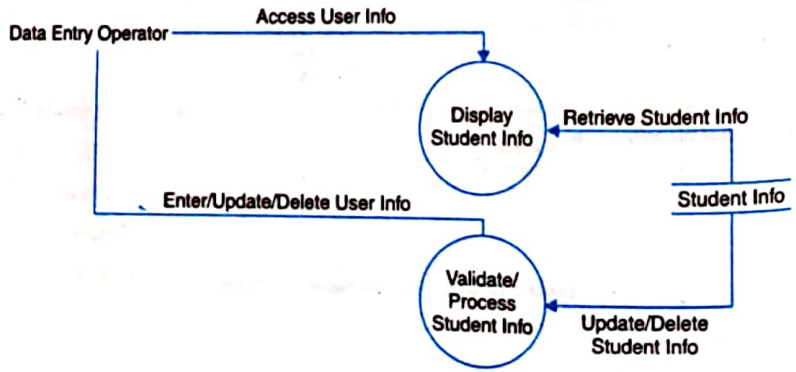
2. Login

The Level 2 DFD of this process is given below:



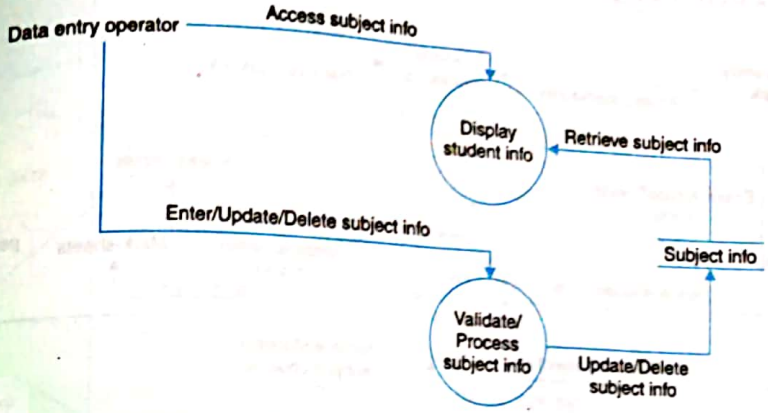
3. Student information management

The Level 2 DFD of this process is given below:



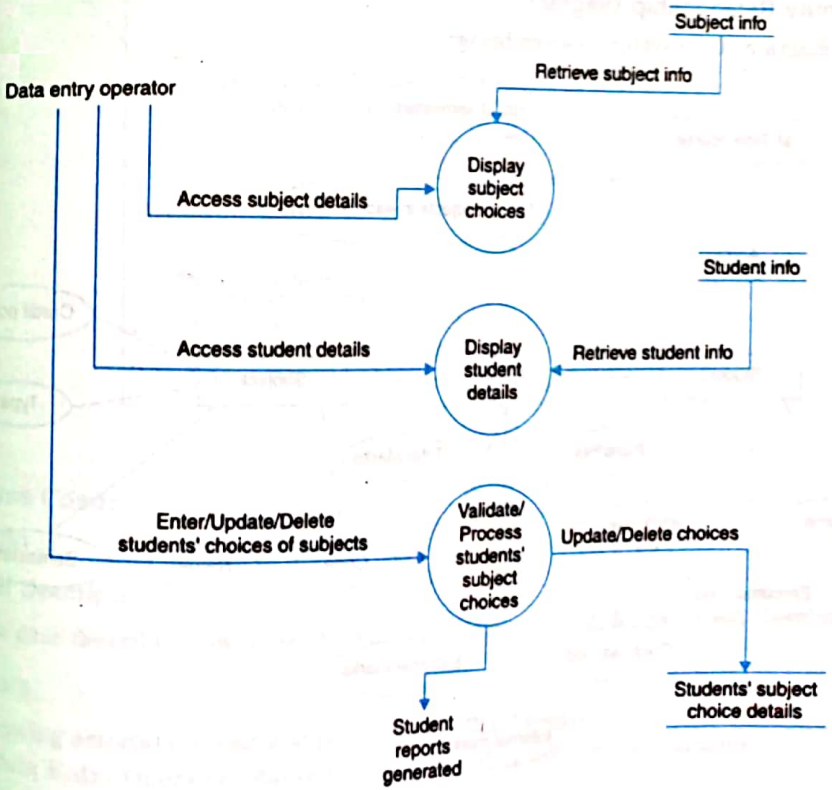
4. Subject information management

The Level 2 DFD of this process is given below:



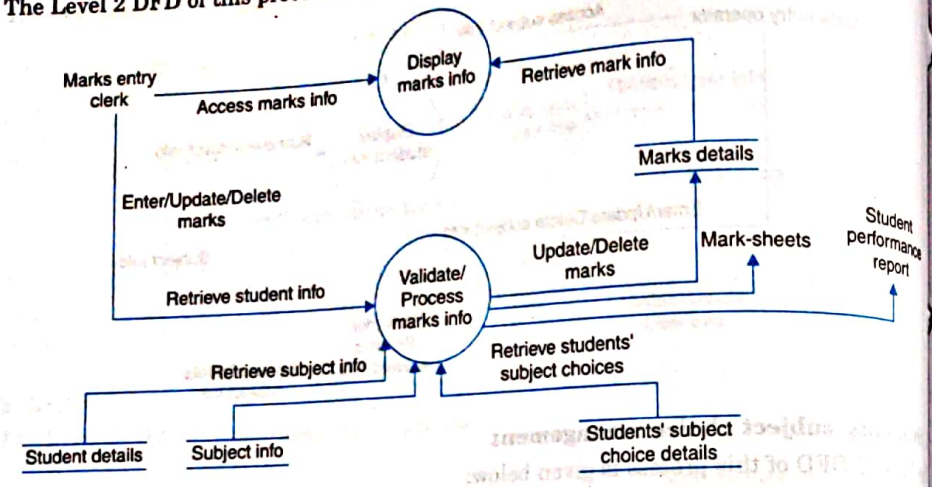
5. Students' subject choice management

The Level 2 DFD of this process is given below:



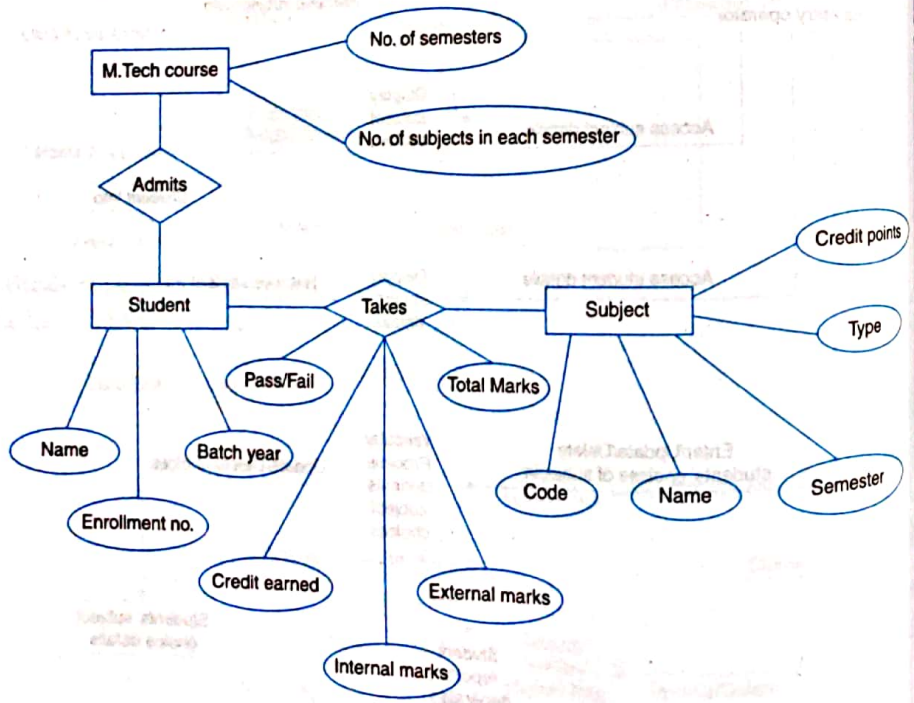
6. Marks information management

The Level 2 DFD of this process is given below:

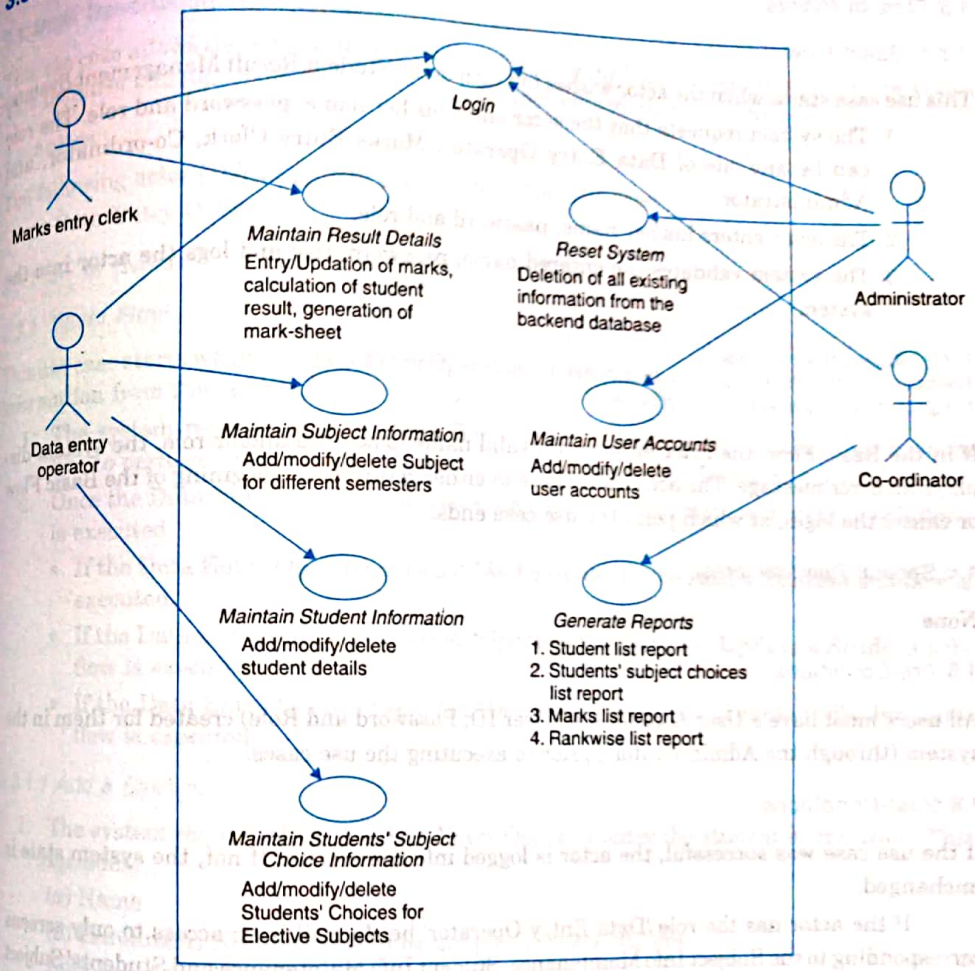


3.9.4 Entity Relationship Diagram

The ER diagram of the system is given below:



3.9.5 Use Case Diagram



Use Case Diagram

3.9.6 Use Cases

1 Login

1.1 Brief Description

This use case describes how a user logs into the Student Result Management System.

1.2 Actors

The following actor(s) interact and participate in this use case:

Data Entry Operator, Marks Entry Clerk, Administrator, Co-ordinator.

1.3 Flow of Events

1.3.1 Basic Flow

This use case starts when the actor wishes to Login to the Student Result Management System.

1. The system requests that the actor enter his/her name, password and role. The role can be any one of Data Entry Operator, Marks Entry Clerk, Co-ordinator, and Administrator.
2. The actor enters his/her name, password and role.
3. The system validates the entered name, password, role and logs the actor into the system.

1.3.2 Alternative Flows

1.3.2.1 Invalid Name/Password/Role

If in the Basic Flow, the actor enters an invalid name, password and/or role, the system displays an error message. The actor can choose to either return to the begining of the Basic Flow or cancel the login, at which point the use case ends.

1.4 Special Requirements

None

1.5 Pre-Conditions

All users must have a User Account (*i.e.*, User ID, Password and Role) created for them in the system (through the Administrator), prior to executing the use cases.

1.6 Post-Conditions

If the use case was successful, the actor is logged into the system. If not, the system state is unchanged.

If the actor has the role 'Data Entry Operator' he/she will have access to only screens corresponding to the Subject Info Maintenance, Student Info Maintenance and Students' Subject Choice Info Maintenance modules of the system.

If the actor has the role 'Marks Entrey Clerk', he/she will have access to only screens corresponding to the Marks Info Maintenance module of the system. If the actor has the role 'Co-ordinator', he/she will only be able to view/print the various reports generated by the system.

If the actor has the role 'Administrator' he/she will have access to only screens corresponding to User Account maintenance module and Reset System feature of the system.

1.7 Extension Points

None

2 Maintain Student Information

2.1 Brief Description

This use case allows the actor with role 'Data Entry Operator' to maintain student information. This includes adding, changing and deleting studnet information from the system.

2.2 Actors

The following actor (*s*) interact and participate in this use case:

Data Entry Operator.

2.3 Flow of Events

2.3.1 Basic Flow

This use case starts when the Data Entry Operator wishes to add, change, and/or delete student information from the system.

1. The system requests that the Data Entry Operator specify the function he/she would like to perform (either Add a Student, Update a Student, or Delete a Student).
2. Once the Data Entry Operator provides the requested information, one of the sub-flows is executed.
   - If the Data Entry Operator selected "Add a Student", the **Add a Student** sub-flow is executed.
   - If the Data Entry Operator selected "Update a Student", the **Update a Student** sub-flow is executed.
   - If the Data Entry Operator selected "Delete a Student", the **Delete a Student** sub-flow is executed.

2.3.1.1 Add a Student

1. The system requests that the Data Entry Operator enter the student information. This includes:

   (*a*) Name

   (*b*) Enrollment Number–should be unique for every student

   (*c*) Year of Enrollment
2. Once the Data Entry Operator provides the requested information, the student is added to the system and an appropriate message is displayed.

2.3.1.2 Update a Student

1. The system requests that the Data Entry Operator enters the student enrollment number.
2. The Data Entry Operator enters the student enrollment number. The system retrieves and displays the student information.
3. The Data Entry Operator makes the desired changes to the student information. This includes any of the information specified in the **Add a Student** sub-flow.
4. Once the Data Entry Operator updates the necessary information, the system updates the student record with the updated information.

2.3.1.3 Delete a Student

1. The system requests that the Data Entry Operator enters the student enrollment number.
2. The Data Entry Operator enters the student enrollment number. The system retrieves and displays the student information.
3. The system prompts the Data Entry Operator to confirm the deletion of the student.
4. The Data Entry Operator confirms the deletion.
5. The system deletes the student record.

2.3.2 Alternative Flows

2.3.2.1 Student Not Found

If in the **Update a Student** or **Delete a Student** sub-flows, a student with the specified enrollment number does not exist, the system displays an error message. The Data Entry Operator can then enter a different enrollment number or cancel the operation, at which point the use case ends.

2.3.2.2 Update Cancelled

If in the **Update a Student** sub-flow, the Data Entry Operator decides not to update the student information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

2.3.2.3 Delete Cancelled

If in the **Delete a Student** sub-flow, the Data Entry Operator decides not to delete the student information, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

2.4 Special Requirements

None

2.5 Pre-Conditions

The Data Entry Operator must be logged onto the system before this use case begins.

2.6 Post-Conditions

If the use case was successful, the student information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

2.7 Extension Points

None

3 Maintain Subject Information

3.1 Brief Description

This use case allows the actor with role 'Data Entry Operator' to maintain subject information. This includes adding, changing and deleting subject information from the system.

3.2 Actors

The following actor (s) interact and participate in this use case:

Data Entry Operator

3.3 Flow of Events

3.3.1 Basic Flow

This use case starts when the Data Entry Operator wishes to add, change, and/or delete subject information from the system.

1. The system requests that the Data Entry Operator specify the function he/she would like to perform (either Add a Subject, Update a Subject, or Delete a Subject).
2. Once the Data Entry Operator provides the requested information, one of the sub-flows is executed.
   - If the Data Entry Operator selected "Add a Subject", the **Add a Subject** sub-flow is executed.
   - If the Data Entry Operator selected "Update a Subject", the **Update a Subject** sub-flow is executed.
   - If the Data Entry Operator selected "Delete a Subject", the **Delete a Subject** sub-flow is executed.

3.3.1.1 Add a Subject

1. The system requests that the Data Entry Operator enters the subject information. This includes:
   (a) Name of the subject
   (b) Subject Code–should be unique for every subject
   (c) Semester
   (d) Subject Type–can be Core 1/Core 2/Dissertation/Elective 1/Elective 2/Lab 1/Lab 2/Minor Project/Seminar/Term Paper.
   (e) Credits.
2. Once the Data Entry Operator provides the requested information, the subject is added to the system and an appropriate message is displayed.

3.3.1.2 Update a Subject

1. The system requests that the Data Entry Operator enters the subject code.
2. The Data Entry Operator enters the subject code. The system retrieves and displays the subject information.
3. The Data Entry Operator makes the desired changes to the subject information. This includes any of the information specified in the **Add a Subject** sub-flow.
4. Once the Data Entry Operator updates the necessary information, the system updates the subject record with the updated information.

2.3.1.3 Delete a Student

1. The system requests that the Data Entry Operator enters the student enrollment number.
2. The Data Entry Operator enters the student enrollment number. The system retrieves and displays the student information.
3. The system prompts the Data Entry Operator to confirm the deletion of the student.
4. The Data Entry Operator confirms the deletion.
5. The system deletes the student record.

2.3.2 Alternative Flows

2.3.2.1 Student Not Found

If in the **Update a Student** or **Delete a Student** sub-flows, a student with the specified enrollment number does not exist, the system displays an error message. The Data Entry Operator can then enter a different enrollment number or cancel the operation, at which point the use case ends.

2.3.2.2 Update Cancelled

If in the **Update a Student** sub-flow, the Data Entry Operator decides not to update the student information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

2.3.2.3 Delete Cancelled

If in the **Delete a Student** sub-flow, the Data Entry Operator decides not to delete the student information, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

2.4 Special Requirements

None

2.5 Pre-Conditions

The Data Entry Operator must be logged onto the system before this use case begins.

2.6 Post-Conditions

If the use case was successful, the student information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

2.7 Extension Points

None

3 Maintain Subject Information

3.1 Brief Description

This use case allows the actor with role 'Data Entry Operator' to maintain subject information. This includes adding, changing and deleting subject information from the system.

3.2 Actors

The following actor (s) interact and participate in this use case:

Data Entry Operator

3.3 Flow of Events

3.3.1 Basic Flow

This use case starts when the Data Entry Operator wishes to add, change, and/or delete subject information from the system.

1. The system requests that the Data Entry Operator specify the function he/she would like to perform (either Add a Subject, Update a Subject, or Delete a Subject).
2. Once the Data Entry Operator provides the requested information, one of the sub-flows is executed.
   - If the Data Entry Operator selected "Add a Subject", the **Add a Subject** sub-flow is executed.
   - If the Data Entry Operator selected "Update a Subject", the **Update a Subject** sub-flow is executed.
   - If the Data Entry Operator selected "Delete a Subject", the **Delete a Subject** sub-flow is executed.

3.2.1.1 Add a Subject

1. The system requests that the Data Entry Operator enters the subject information. This includes:
   (*a*) Name of the subject
   (*b*) Subject Code–should be unique for every subject
   (*c*) Semester
   (*d*) Subject Type–can be Core 1/Core 2/Dissertation/Elective 1/Elective 2/Lab 1/Lab 2/ Minor Project/Seminar/Term Paper.
   (*e*) Credits.
2. Once the Data Entry Operator provides the requested information, the subject is added to the system and an appropriate message is displayed.

3.3.1.2 Update a Subject

1. The system requests that the Data Entry Operator enters the subject code.
2. The Data Entry Operator enters the subject code. The system retrieves and displays the subject information.
3. The Data Entry Operator makes the desired changes to the subject information. This includes any of the information specified in the **Add a Subject** sub-flow.
4. Once the Data Entry Operator updates the necessary information, the system updates the subject record with the updated information.

3.3.1.3 Delete a Subject

1. The system requests that the Data Entry Operator enter the subject code.
2. The Data Entry Operator enters the subjects code. The system retrieves and displays the subject information.
3. The system prompts the Data Entry operator to confirm the deletion of the subject.
4. The Date Entry Operator confirms the deletion.
5. The system deletes the subject record.

3.3.2 Alternative Flows

3.3.2.1 Subject Not Found

If in the **Update a Subject** or **Delete a Subject** sub-flows, a subject with the specified subject code does not exist, the system displays an error message. The Data Entry Operator can then enter a different subject code or cancel the operation, at which point the use case ends.

3.3.2.2 Update Cancelled

If in the **Update a Subject** sub-flow, the Data Entry Operator decides not to update the subject information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

3.3.2.3 Delete Cancelled

If in the **Delete a Subject** sub-flow, the Data Entry Operator decides not to delete the subject information, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

3.4 Special Requirements

None

3.5 Pre-Conditions

The Data Entry Operator must be logged onto the system before this use case begins.

3.6 Post–Conditions

If the use case was successful, the student information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

3.7 Extension Points

None

4 Maintain Students' Subject Choice Infromation

4.1 Brief Description

This use case allows the actor with role 'Data Entry Operator' to maintain information about the choice of different Elective subjects opted by various students. This includes displaying the various available choices of Elective subjects available during a particular semester and updating the information about the choice of Elective Subject (s) opted by different students of that semester.

4.2 Actors

The following actor (s) interact and participate in this use case:

Data Entry Operator

4.3 Flow of Events

4.3.1 Basic Flow

This use case starts when the Data Entry Operator wishes to update students' Subject Choice information from the system.

1. The system requests that the Data Entry Operator specify the semester and enrollment year of students, for which the Students' Subject Choices have to be updated.
2. Once the Data Entry Operator provides the requested information, the system displays the list of available choices for Elective I and Elective II subjects for that semester and the list of students enrolled in the given enrollment year (alongwith their existing subject choices, if any).
3. The system requests that the Data Entry Operator specify the information regarding Students' Subject Choices. this includes
   (a) Student's Enrollment Number
   (b) Student's Choice for Elective I subject (the corresponding subject code)
   (c) Student's Choice for Elective II subject (the corresponding subject code).
4. Once the Data Entry Operator provides the requested information, the information regarding Student's Subject Choices is added/updated in the system and an appropriate message is displayed.

4.3.2 Alternative Flows

4.3.2.1 Subject Information Does Not Exist

If no or incomplete subject information exists in the system for the semester specified by the Data Entry Operator, the system displays an error message. The Data Entry Operator can then enter a different semester or cancel the operation, at which point the use case ends.

4.3.2.2 Student Information Does Not Exist

If no student information exists in the system for the enrollment year specified by the Data entry Operator, the system displays an error message. The Data Entry Operator can then enter a different enrollment year or cancel the operation, at which point the use case ends.

4.3.2.3 Incorrect Choice Entered for Elective I/Elective II Subjects

If the subject code entered by the Data Entry Operator for Elective I/Elective II subject does not exist in the system, the system displays an error message.

The Data Entry Operator can then enter the correct subject code or cancel the operation, at which point the use case ends.

4.3.2.4 Update Cancelled

If in the **Basic Flow**, the Data Entry Operator decides not to update the subject information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

4.4 Special Requirements

None

4.5 Pre-Conditions

The Data Entry Operator must be logged onto the system before this use case begins.

4.6 Post-Conditions

If the use case was successful, information about students' choices for opting different Elective Subjects is added/updated in the system. Otherwise, the system state is unchanged.

4.7 Extension Points

None

5 Maintain Result Details

5.1 Brief Description

This use case allows the actor with role 'Marks Entry Clerk' to maintain subject-wise marks information of each student, in different semesters. This includes adding, changing and deleting marks information from the system.

5.2 Actors

The following actor (s) interact and participate in this use case:
Maks Entry Clerk.

5.3 Flow of Events

5.3.1 Basic Flow

This use case starts when the Marks Entry Clerk wishes to add, change, and/or delete marks information from the system.

1. The system requests that the Marks Entry Clerk specify the function he/she would like to perform (either Add Marks, Update Marks, Delete Marks, or Generate Mark-sheet).
2. Once the Marks Entry Clerk provides the requested information, one of the sub-flows is executed.
   - If the Marks Entry Clerk selected "Add Marks", the **Add Marks** sub-flow is executed.
   - If the Marks Entry Clerk selected "Update Marks", the **Update Marks** sub-flow is executed.
   - If the Marks Entry Clerk selected "Delete Marks", the **Delete Marks** sub-flow is executed.
   - If the Marks Entry Clerk selected "Generate Mark-sheet", the **Generate Mark-sheet** sub-flow is executed.

5.3.1.1 Add Marks Record

1. The system requests that the Marks Entry Clerk enters the marks information. This includes:

   (a) Selecting a semester
   (b) Selecting a subject code
   (c) Selecting the student enrollment number
   (d) Entering the internal/external marks for that semester, subject code and enrollment number.
2. Once the Marks Entry Clerk provides the requested information, the system saves the marks and an appropriate message is displayed.

5.3.1.2 Update Marks Record

1. The system requests the Marks Entry Clerk to make following entries:
   (a) Selecting the semester
   (b) Selecting the subject code for which marks have to be updated
   (c) Selecting the student enrollment number.
2. Once the Marks Entry Clerk provides the requested information, the system retrieves and displays the corresponding marks details.
3. The Marks Entry Clerk makes the desired changes to the internal/external marks details.
4. The system updates the marks record with the changed information.

5.3.1.3 Delete Marks Record

1. The system requests the Marks Entry Clerk to make following entries:
   (a) Selecting the semester
   (b) Selecting the subject code for which marks have to be updated
   (c) Selecting the student enrollment number.
2. Once the Marks Entry Clerk provides the requested information, the system retrievs and displays the corresponding marks record from the database.
3. The system verifies if the Marks Entry Clerk wishes to proceed with the deletion of the record. Upon confirmation, the record is deleted from the system.

5.3.1.4 Compute Result

1. Once all the marks are added to the database, the result is computed for each student.
2. If the student has scored more than 50% in a subject, the associated credit points are allotted to that student.
3. The average percentage marks are claculated for the student and his/her division is also derived based on the percentage.

5.3.1.5 Generate Mark-Sheet

1. The system requests that the Marks Entry Clerk specify the Enrollment Number of the student and the semester for which mark-sheet is to be generated.
2. Once the Marks Entry Clerk provides the requested information, the system generates a printable mark-sheet for the specified student and displays it.
3. The Marks Entry Clerk can then issue a print request for the mark-sheet to be printed.

5.3.2 Alternative Flows

5.3.2.1 Record Not Found

If in the **Update Marks, Delete Marks** or **Generate Mark-sheet** sub-flows, a record with the specified information does not exist, the system displays an error message. The Marks Entry Clerk can then enter different information for retrieving the record or cancel the operation, at which point the use case ends.

5.3.2.2 Update Cancelled

If in the **Update Marks** sub-flow, the Marks Entry Clerk decides not to update the marks, the update is cancelled and the **Basic Flow** is re-started at the beginning.

5.3.2.3 Delete Cancelled

If in the **Delete Marks** sub-flow, the Marks Entry Clerk decides not to delete the marks, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

5.4 Special Requirements

None

5.5 Pre-Conditions

The Marks Entry Clerk must be logged onto the system before this use case begins.

5.6 Post–Conditions

If the use case was successful, the marks information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

5.7 Extension Points

None

6 Generate Reports

6.1 Brief Description

This use case allows the actor with role 'Co-ordinator' to generate various reports. The following reports can be generated:

(*a*) Student List Report

(*b*) Students' Subject Choices List Report

(*c*) Marks List Report

(*d*) Rank-wise List Report

6.2 Actors

The following actor(s) interact and participate in this use case:

Co-ordinator

6.3 Flow of Events

6.3.1 Basic Flow

This use case starts when the Co-ordinator wishes to generate reports.

1. The system requests the Co-ordinator specify the report he/she would like to generate.
2. Once the Co-ordinator provides the requested information, one of the sub-flows is executed:
   - If the Co-ordinator selected "Student List Report", the **Generate Student List Report** sub-flow is executed.
   - If the Co-ordinator selected "Students' Subject Choices List Report", the **Generate Students' Subject Choices List Report** sub-flow is executed.
   - If the Co-ordinator selected "Marks List Report", the **Generate Marks List Report** sub-flow is executed.
   - If the Co-ordinator selected "Rank-wise List Report", the **Generate Rank-wise List Report** sub-flow is executed.

6.3.1.1 Generate Student List Report

1. The system requests that the Co-ordinator provide the enrollment year for which the Student List report is to be generated.
2. Once the Co-ordinator provides the requested infromation, the system generates the Student List report, containing the list of students enrolled in the given year.
3. The Co-ordinator can then issue a print request for the report to be printed.

6.3.1.2 Generate Student's Subject Choices List Report

1. The system requests that the Co-ordinator provides the enrollment year and the semester for which the Students' Subject Choices List report is to be generated.
2. Once the Co-ordinator provides the requested information, the system generates the Students' subject Choices List report, containing the choices for Elective I and Elective II subjects, opted by the students of the given enrollment year and semester.
3. The Co-ordinator can then issue a print request for the report to be printed.

6.3.1.3 Generate Marks List Report

1. The system requests that the Co-ordinator provides the enrollment year and the semester for which the Marks List report is to be generated.
2. Once the Co-ordinator provides the requested information, the system generates the Marks List report, containing the marks details of various students in all the subjects for the given enrollment year and semester.
3. The Co-ordinator can then issue a print request for the report to be printed.

6.3.1.4 Generate Rank-wise List Report

1. The system requests that the Co-ordinator provide the enrollment year and the semester for which the Rank-wise List report is to be generated.

2. Once the Co-ordinator provides the requested information, the system generates the Rank-wise List report, containing the percentage wise and rank-wise list of all students (alongwith their total marks and division) for the given enrollment year and semester.
3. The Co-ordinator can then issue a print request for the report to be printed.

6.3.2 Alternative Flows

6.3.2.1 Student Not Found

If no student information exists in the system for the enrollment year specified by the Co-ordinator, the system displays an error message. The Co-ordinator can then enter a different enrollment year or cancel the operation, at which point the use case ends.

6.4 Special Requirements

None

6.5 Pre-Conditions

The Co-ordinator must be logged onto the system before this use case begins.

6.6 Post–Conditions

If the use case was successful, the desired report is generated. Otherwise, the system state is unchanged.

6.7 Extension Points

None.

7 Maintain User Accounts

7.1 Brief Description

This use case allows the actor with role 'Administrator' to maintain User Account. This includes adding, changing and deleting user account information from the system.

7.2 Actors

The following actor (s) interact and participate in this use case:
Administrator.

7.3 Flow of Events

7.3.1 Basic Flow

This use case starts when the Administrator wishes to add, change, and/or delete use account information from the system.

1. The system requests that the Administrator specify the function he/she would like to perform (either Add a User Account, Update a User Account, or Delete a User Account).
2. Once the Administrator provides the requested information, one of the sub-flows is executed.

- If the Administrator selected "Add a User Account", the **Add a User Account** sub-flow is executed.
- If the Administrator selected "Update a User Account", the **Update a User Account** sub-flow is executed.
- If the Administrator selected "Delete a User Account", the **Delete a User Account** sub-flow is executed.

7.3.1.1 Add a User Account

1. The system requests that the Administrator enters the user information. This includes:
   (a) User Name
   (b) User ID-should be unique for each user account
   (c) Password
   (d) Role
2. Once the Administrator provides the requested information, the user account information is added to the system and an appropriate message is displayed.

7.3.1.2 Update a User Account

1. The system requests that the Administrator enters the User ID.
2. The Administrator enters the User ID. The system retrieves and displays the user account information.
3. The Adiministrator makes the desired changes to the user account information. This includes any of the information specified in the **Add a User Account** sub-flow.
4. Once the Administrator updates the necessary information, the system updates the user account record with the updated information.

7.3.1.3 Delete a User Account

1. The system requests that the Administrator enters the User ID.
2. The Administrator enters the User ID. The system retrieves and displays the user account information.
3. The system prompts the Administrator to confirm the deletion of the user account.
4. The Administrator confirms the deletion.
5. The system deletes the user account record.

7.3.2 Alternative Flows

7.3.2.1 User Not Found

If in the **Update a User Account** or **Delete a User Account** sub-flows, a user account with the specified User ID does not exist, the system displays an error message. The Administrator can then enter a different User ID or cancel the operation, at which point the use case ends.

7.3.2.2 Update Cancelled

If in the **Update a User Account** sub-flow, the Administrator decides not to update the user account information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

7.3.2.3 Delete Cancelled

If in the **Delete a User Account** sub-flow, the Administrator decides not to delete the user account information, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

7.4 Special Requirements

None

7.5 Pre-Conditions

The Administrator must be logged onto the system before this use case begins.

7.6 Post-Conditions

If the use case was successful, the user account information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

7.7 Extension Points

None

8. Reset System

8.1 Brief Description

This Use case allows the actor with role 'Administrator' to reset the system by deleting all existing information from the system.

8.2 Actors

The following actor (s) interact and participate in this use case:

Adiministrator

8.3 Flow of Events

8.3.1 Basic Flow

This use case starts when the Administrator wishes to reset the system.

1. The system requests the Administrator to confirm if he/she wants to delete all the existing information from the system.
2. Once the Administrator provides confirmation, the system deletes all the existing information from the backend database and displays an appropriate message.

8.3.2 Alternative Flows

8.3.2.1 Reset Cancelled

If in the **Basic Flow,** the Administrator decides not to delete the entire existing information, the reset is cancelled and the use case ends.

8.4 Special Requirements

None

8.5 Pre-Conditions

The Administrator must be logged onto the system before this use case begins.

8.6 Post-Conditions

If the use case was successful, all the existing information is deleted from the backend database of the system. Otherwise, the system state is unchanged.

8.7 Extension Points

None

3.9.7 SRS Document

1 Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, Acronyms, and Abbreviations
- 1.4 References
- 1.5 Overview

2 Overall Description
- 2.1 Product Perspective
  - 2.1.1 System Interfaces
  - 2.1.2 User Interfaces
  - 2.1.3 Hardware Interfaces
  - 2.1.4 Software Interfaces
  - 2.1.5 Communications Interfaces
  - 2.1.6 Memory Constraints
  - 2.1.7 Operations
  - 2.1.8 Site Adaptation Requirements
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Assumptions and Dependencies
- 2.6 Apportioning of Requirements

3 Specific Requirements
- 3.1 External Interface Requirements
  - 3.1.1 User Interfaces
    - Login Screen:
    - Subject Info Parameters Screen:
    - Subject Information Screen:
    - Student Info Parameters Screen:
    - Student Information Screen:
    - Students' Subject Choice Parameters Screen:
    - Students' Subject Choice Information Screen:

(Contd.)...

Marks Entry Parameters Screen:
Marks Entry Screen:
Mark-sheet Parameters Screen:
Students List Report Parameters Screen:
Marks List Report Parameters Screen:
Rank-wise List Report Parameters Screen:
Students' Subject Choices List Report Parameters Screen:

3.1.2 Hardware Interfaces
3.1.3 Software Interfaces
3.1.4 Communications Interfaces

3.2 Software Product Features
3.2.1 Subject Information Maintenance
Validity Checks
Sequencing Information
Error Handling/ Response to Abnormal situations
3.2.2 Student Information Maintenance
Validity Checks
Sequencing Information
Error Handling / Response to Abnormal situations
3.2.3 Marks Info Maintenance
Validity Checks
Sequencing Information
Error Handling/ Response to Abnormal situations
3.2.4 Mark sheet Generation
3.2.5 Report Generation
Student List Reports
Students' Subject Choices List Reports
Semester-wise Mark lists
Rank-wise List Report

3.3 Performance Requirements
3.4 Design Constraints
3.5 Software System Attributes
3.5.1 Security
3.5.2 Maintainability
3.5.3 Portability
3.6 Logical Database Requirements
3.7 Other Requirements

1 Introduction

This document aims at defining the overall software requirements for 'Student Res Management System'. Efforts have been made to define the requirements exhaustively a

accurately. The final product will be having only features/functionalities mentioned in this document and assumptions for any additional functionality/feature should not be made by any of the parties involved in developing/testing/implementing/using this product. In case it is required to have some additional features, a formal change request will need to be raised and subsequently a new release of this document and/or product will be produced.

1.1 Purpose

This specification document describes the capabilities that will be provided by the software application 'Student Result Management System'. It also states the various required constraints by which the system will abide. The intended audience for this document are the development team, testing team and end users of the product.

1.2 Scope

The software product 'Student Result Management System' will be an MIS and Reporting application that will be used for result preparation and management of M. Tech. Program of a University. The application will manage the information about various students enrolled in this course in different years, the subjects offered during different semesters of the course, the students' choices for opting different subjects, and the marks obtained by various students in various subjects in different semesters. Printable reports regarding list of students, marks obtained by all students in a particular semester and performance of students (rank-wise, percentage-wise, pass/fail, division-wise.) will be generated. The system will also generate printable mark-sheets for individual students.

The application will greatly simplify and speed up the result preparation and management process.

1.3 Definitions, Acronyms, and Abbreviations

Following abbreviations have been used throughout this document:

M.Tech: Master of Technology
IT: Information Technology
DBA: Database Administrator

1.4 References

(i) *University website:* For information about course structure of M.Tech. Program
(ii) IEEE Recommended Practice for Software Requirements Specifications–IEEE Std 830-1993

1.5 Overview

The rest of this SRS document describes the various system requirements, interfaces, features and functionalities in detail.

2 Overall Description

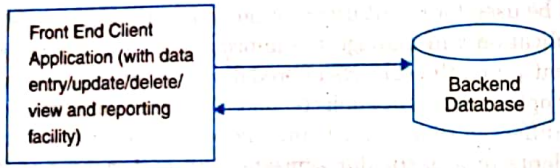
M.Tech. Program is a 4-semester course. The students are offered 4 subjects (theory) and 2 Labs (practical) during first, second and third semesters. Students also have to submit a term paper/minor project in 2nd and 3rd semesters. The fourth semester consists of a seminar and

disseratation. Each subject/lab/term paper/seminar/dissertation has credits associated with it. When a student secures pass marks in a paper he/she also earns all the credit (s) assigned to that paper.

The 'Student Result Management System' will have capability to maintain information about students enrolled in the course, the subjects offered to students during different semesters, the students' choices for opting different Elective subjects (out of the available ones) and the marks obtained by students in different subjects in various semesters. The software will also generate summary reports regarding student information, semester-wise mark lists and performance reports. Printable mark-sheets of individual students will also be generated by the application.

2.1 Product Perspective

The application will be a windows-based, self-contained and independent software product.



2.1.1 System Interfaces

None

2.1.2 User Interfaces

The application will have a user-friendly and menu based interface. Following screens will be provided:

(*i*) A Login screen for entering the username, password and role (Administrator, Data Entry Operator, Marks Entry Clerk, Co-ordinator) will be provided. Access to different screens will be based upon the role of the user.

(*ii*) There will be a screen for capturing and displaying information regarding what all subjects are offered during which semester, how many credit points are assigned to that subject and whether the subject is an elective, a core paper, a lab paper, a term paper or a dissertation.

(*iii*) There will be a screen for capturing and displaying information regarding various students enrolled for the course in different years.

(*iv*) There will be a screen for capturing and displaying information regarding which student is currently enrolled in which semester and what all elective subjects he/she has opted.

(*v*) There will be a screen that will capture information regarding which student has scored how many marks (internal + external evaluation) in each subject (in a particular semester). Credits in each subject will be calculated depending upon the marks obtained in that subject.

(*vi*) There will be a screen for capturing and displaying information regarding which all user accounts exist in the system, thus showing who all can access the system.

The following reports will be generated:

(*i*) *Students' List Report:* Printable reports will be generated to show the list of students enrolled in a particular batch year.

(*ii*) *Students' Subject Choices List Report:* For Ist, IInd and IIIrd semester, there will be printable reports showing the different elective subjects opted by various students (enrolled in a particular batch year) of the corresponding semester.

(*iii*) *Marks List Report:* For each semester there will be a printable report showing the subject-wise marks details for all students of that semester.

(*iv*) *Rank-wise List Report:* For each semester there will be a printable report showing the percentage-wise and rank-wise list of students alongwith the division secured.

(*v*) *Mark-sheet:* For each student of each semester, a printable mark-sheet will be generated, showing the subject-wise marks details, Total marks, total credits, Percentage, Pass/Fail status for that student.

2.1.3 Hardware Interfaces

(*i*) Screen resolution of at least 800 × 600–required for proper and complete viewing of screens. Higher resolution would not be a problem.

(*ii*) Support for printer (dot-matrix/deskJet/inkjet etc.–any will do)–that is, appropriate drivers are installed and printer connected printer will be required for printing of reports and mark-sheets.

(*iii*) Standalone system or network based–not a concern, as it will be possible to run the application on any of these.

2.1.4 Software Interfaces

(*i*) Any windows-based operating system (Windows 95/98/2000/XP/NT)

(*ii*) MS Access 2000 as the DBMS—for database. Future release of the application will aim at upgrading to Oracle 8*i* as the DBMS.

(*iii*) Crystal Reports 8—for generating and viewing reports.

(*iv*) Visual Basic 6—for coding/developing the software.

Software mentioned in pts. (*iii*) and (*iv*) above, will be required only for development of the application. The final application will be packaged as an independent setup program that will be delivered to the client (University in this case).

2.1.5 Communications Interfaces

None

2.1.6 Memory Constraints

At least 64 MB RAM and 2 GB space on hard disk will be required for running the application.

2.1.7 Operations

This product release will not cover any automated housekeeping aspects of the database. The DBA at the client site (*i.e.*, University) will be responsible for manually deleting old/non-required data. Database backup and recovery will also have to be handled by the DBA. However, the system will provide a 'RESET SYSTEM' function that will delete (upon confirmation from the Administrator) all the existing information from the database.

2.1.8 Site Adaptation Requirements

The terminals at client site will have to support the hardware and software interfaces specified in above sections.

2.2 Product Functions

The system will allow access only to authorised users with specific roles (Administrator, Data Entry Operator, Marks Entry Clerk and Co-ordinator). Depending upon the user's role, he/she will be able to access only specific modules of the system.

A summary of the major functions that the software will perform:

(*i*) A Login facility for enabling only authorised access to the system.

(*ii*) User (with role Data Entry Operator) will be able to add/modify/delete information about different students that are enrolled for the course in different years.

(*iii*) User (with role Data Entry Operator) will be able to add/modify/delete information about different subjects that are offered in a particular semester. The semester-wise list of subjects along with their credit points and type (*i.e.*, elective/core/lab/term paper/dissertation) will also be displayed.

(*iv*) User (with role Data Entry Operator) will be able to add/modify/delete information about the Elective subjects opted by different students in different semesters.

(*v*) User (with role Marks Entry Clerk) will be able to add/modify/delete information regarding marks obtained by different students in different semesters.

(*vi*) User (with role Marks Entry Clerk) will also be able to print mark-sheets of students.

(*vii*) User (with role Co-ordinator) will be able to generate Printable reports (as mentioned in section 2.1.2 above).

(*viii*) User (with role Administrator) will be able to 'Reset' the system–leading to deletion of all existing information from the backend database.

(*ix*) User (with role Administrator) will be able to create/modify/delete new/existing user accounts.

2.3 User Characteristics

- *Educational level:* At least graduate should be comfortable with English language.
- *Experience:* Should be well versed/informed about the course structure of M. Tech program of University. Entry of marks or their modification can be done only by user who is authorised for this job by the result preparation committee of University.

- *Technical expertise:* Should be comfortable using general-purpose applications on a computer.

2.4 Constraints

(*i*) Since the DBMS being used is MS Access 2000, which is not a very powerful DBMS, it will not be able to store a very huge number of records.

(*ii*) Due to limited features of DBMS being used performance tuning features will not be applied to the queries and thus the system may become slow with the increase in number of records being stored.

(*iii*) Due to limited features of DBMS being used, database auditing will also not be provided.

(*iv*) Users at University will have to implement a security policy to safeguard the marks-related information from being modified by unauthorised users (by means of gaining access to the backend database).

2.5 Assumptions and Dependencies

(*i*) The number of subjects to be taken up by a student in each semester does not change.

(*ii*) The subject types (*i.e.*, elective, core, lab, term paper and dissertation) do not change.

(*iii*) The number of semesters in the M. Tech. Program does not change.

2.6 Apportioning of Requirements

Not Required.

3 Specific Requirements

This section contains the software requirements to a level of detail sufficient to enable designers to design the system, and testers to test that system.

3.1 External Interface Requirements

3.1.1 User Interfaces

The following screens will be provided:

Login screen:

This will be the first screen that will be displayed. It will allow user to access different screens based upon the user's role. Various fields available on this screen will be

(*i*) *User ID:* Alphanumeric of length upto 10 characters

(*ii*) *Password:* Alphanumeric of length upto 8 characters

(*iii*) Role: Will have the following values:

Administrator, Marks Entry Clerk, Co-ordinator, Data Entry Operator

Subject info parameters screen:

This screen will be accessible only to user with role Administrator. It will allow the user to enter the semester number for which the user wants to access the subject information.

Subject information screen:

This screen will be accessible only to user with role Administrator. It will allow user to add/modify/delete information about new/existing subject (s) for the semester that was selected in the 'Subject Info Parameters' screen. The list of available subjects for that semester, will also be displayed. Various fields available on this screen will be:

(*i*) *Subject Code:* of the format IT-### (# represents a digit)

(*ii*) *Subject Name:* Alphanumeric, of length upto 50 characters

(*iii*) *Category/Type:* Will have any of the following values:-

Elective 1/Elective 2/Core/Lab/Term paper/Seminar/Dissertation

(*iv*) *Credits:* Numeric, will have any value from 0 to 20.

Student info parameters screen:

This screen will be accessible only to user with role Administrator. It will allow the user to enter the Batch Year for which the user wants to access the student information.

Student information screen:

This screen will be accessible only to user role Administrator. It will allow the user to add/modify/delete information about new/existing student (s) for a particular Batch Year. Batch Year-wise list of students will also be displayed. Various fields available on these screens will be:

(*i*) *Student Enrollment No:* of the format ##/M.Tech. (IT)/YYYY (# represents a digit and YYYY represents the batch year).

(*ii*) *Student Name:* will have only alphabetic letters and length upto 40 characters.

(*iii*) *Batch Year:* of the format YYYY (representing the year in which the student enrolled for the course).

Students' subject choice parameters screen:

This screen will be accessible only to user with role Administrator. It will allow the user to enter the Batch Year and the semester number for which the user wants to access the students' subject choice information.

Students' subject choice information screen:

This screen will be accessible only to user with role Administrator. It will allow user to add/modify/delete students' choices for elective subjects of the semester and batch year selected in "Students' Subject Choice Parameters" screen. For the selected semester it will display the list of available choices for Elective I and for Elective II. The screen will display the list of students enrolled during the selected batch year and currently studying in the selected semester and the user will be able to view/add/modify/delete the subject choices for each student in the list.

Marks entry parameters screen:

This screen will be accessible only to user with role Marks Entry Clerk. It will allow the user to enter the Batch Year, the semester number and the Subject for which the user wants to access the marks information.

Marks entry screen:

This screen will be accessible only to user with role Marks Entry Clerk. It will allow user to add/modify/delete information about marks obtained in the selected subjects by different students of that semester who were enrolled in the Batch Year selected in the 'Marks Entry Parameters' Screen. The screen will display the list of students enrolled during the selected batch year and currently studying the selected subject in the selected semester and the user will be able to view/add/modify/delete the marks for each student in the list.

Various fields available on this screens will be:

(*i*) *Student Enrollment No.:* will display the enrollment numbers of all students of the selected Batch Year studying the selected subject in the selected semester.

(*ii*) *Student Name:* will display the name of the student

(*iii*) *Internal Marks:* between 0 and 40

(*iv*) *External Marks:* between 0 and 60

(*v*) *Total Marks:* sum of Internal Marks and External Marks

Mark-sheet parameters screen:

This screen will be accessible only to user with role Marks Entry Clerk. It will allow the user to enter the Enrollment Number and the semester number of the student for whom the user wants to view/print the mark-sheet.

Students list report parameters screen:

This screen will be accessible only to user with role Co-ordinator. It will allow the user to enter the Batch Year for which the user wants to view/print the students list report.

Marks list report parameters screen:

This screen will be accessible only to user with role Co-ordinator. It will allow the user to enter the Batch Year and the semester for which the user wants to view/print the marks list report.

Rank-wise list report parameters screen:

This screen will be accessible only to user with role Co-ordinator. It will allow the user to enter the Batch Year and the semester for which the user wants to view/print the rank-wise list report.

Students' subject choices list report parameters screen:

This screen will be accessible only to user with role Co-ordinator. It will allow the user to enter the Batch Year and the semester for which the user wants to view/print the students' subject choices list report.

3.1.2 Hardware Interfaces

As stated in Section 2.1.3.

3.1.3 Software Interfaces

As stated in section 2.1.4.

3.1.4 Communications Interfaces

None

3.2 System Features

3.2.1 Subject Information Maintenance

Description

The system will maintain information about various subjects being offered during different semesters of the course. The following information would be maintained for each subject: Subject code, Subject name, Subject Type (Core/Elective 1/Elective 2/Lab 1/Lab 2/Term Paper/Minor Project/Dissertation/Seminar), Semester, Credits.

The system will allow creation/modification/deletion of new/existing subjects and also have the ability to list all the available subjects for a particular semester.

Validity checks

(*i*) Only user with role Data Entry Operator will be authorised to access the Subject Information Maintenance module.

(*ii*) Ist, IInd and IIIrd semesters will have 2 core papers, 2 Elective papers, 2 Lab papers and 1 term paper/Minor Project.

(*iii*) Ist, IInd and IIIrd semesters will have 3 choices (subjects) each of type Elective 1 and of type Elective 2.

(*iv*) IVth semester will have only 1 dissertation and 1 seminar.

(*v*) No two semesters will have the same subject *i.e.*, A subject will be offered only in a particular semester.

(*vi*) Subject code will be unique for every subject.

(*vii*) subject code cannot be blank.

(*viii*) Subject name cannot be blank.

(*ix*) Credits cannot be blank.

(*x*) Credits can have value only between 0 and 20.

(*xi*) Subject Type cannot be blank.

(*xii*) Semester cannot be blank.

Sequencing information

Subject info for a particular semester will have to be entered in the system before any student marks information for that semester can be entered.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true, appropriate error messages will be prompted to the user for doing the needful.

3.2.2 Student Information Maintenance

Description

The system will maintain information about choice of different Elective subjects opted by various students of different enrollment years in different semesters. The following information would be maintained:

Student Enrollment number, Semester, Student's Choice for Elective 1 subject Student's Choice for Elective 2 subject.

The system will allow creation/modification/deletion of new/existing students and also have the ability to list all the students enrolled in a particular year.

Validity checks

(*i*) Only user with role Data Entry Operator will be authorised to access the Student Information Maintenance module.

(*ii*) Every student will have a unique Enrollment Number.

(*iii*) Enrollment Number cannot be blank.

(*iv*) Student name cannot be blank.

(*v*) Enrollment Year cannot be blank.

Sequencing information

Student Info for a particular student will have to be entered in the system before any marks info can be entered for that student.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true, appropriate error messages will be prompted to the user for doing the needful.

3.2.3 Students' Subject Choices Information Maintenance

Description

The system will maintain information about choice of different Elective subjects opted by various students of different enrollment years in different semesters. The following information would be maintained:

Student Enrollment number, Semester, Student's Choice for Elective 1 subject, Student's Choice for Elective 2 subject.

The system will allow creation/modification/deletion of students' subject choices and also have the ability to list all the available students' subject choices for a particular semester.

Validity checks

(*i*) Only user with role Data Entry Operator will be authorised to access the Students' Subject Choices Information Maintenance module.

(*ii*) The subject choice for Elective 1 and Elective 2 can be made only from the list of available choices for that semester.

Sequencing information

Students' Subject Choices Info for a particular student can be entered in the system only after Subject Info has been entered in the system for the given semester and the Student Info for that student has been entered in the system.

Students' Subject Choices Info for a particular student will have to be entered in the system before any marks info can be entered for that student in the given semester.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true appropriate error messages will be prompted to the user for doing the needful.

3.2.4 Marks Information Maintenance

Description

The system will maintain information about marks obtained by various students of different enrollment years in different semesters. The following information would be maintained:

Student Enrollment Number, Semester, Subject Code, Internal Marks, External Marks, Total Marks, and Credits.

The system will allow creation/modification/deletion of marks information and also have the ability to list all the available marks information for all students for a particular subject in the given semester.

Validity checks

(*i*) Only user with role Marks Entry Clerk will be authorised to access the Marks Information Maintenance module.

(*ii*) Internal Marks for any subject cannot be less than 0 and greater than 40.

(*iii*) External marks for any subject cannot be less than 0 and greater than 60.

(*iv*) Total marks in any subject will be calculated as: Internal Marks in that subject + External Marks in that subject.

(*v*) If the total Marks in a subject are > = 50, all the credit points associated with the subject will be given to the student, else the credit points earned by the student will be 0 for that subject.

Sequencing information

Marks Info for a particular student can be entered in the system only after Subject Info has been entered in the system for the given semester, the Student Info for that student has been entered in the system, and the Students' Subject Choice Info has been entered in the system for that student in the given semester.

Marks info for a particular student will have to be entered in the system before the student's mark-sheet can be generated.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true appropriate error messages will be prompted to the user for doing the needful.

3.2.5 Mark-sheet Generation

Description

The system will generate mark-sheet for every student in different semesters.

Mark-sheet will have the following format:

Name of the University
Name of Program
Semester <no.>
Mark-sheet

Student Enrollment No._____ Student Name: _____

S.No.	Subject	Internal Marks (Out of 40)	External Marks (Out of 60)	Total Marks (Int. + Ext.)	Pass/Fail	Credits Earned
1.						
2.						
3.						
4.						
5.						
6.						

Marks Grand Total: _____/600 Total Credits:_____

Result: (Pass/Fail)

Date: Signature of Controller of Examinations

There will be a 'Print' icon at the top of mark-sheet for printing the mark-sheet.

Validity checks

(*i*) Only user with role Marks Entry Clerk will be authorised to access the Mark-sheet Generation module.

Sequencing information

Marks-sheet for a particular student can be generated by the system only after Subject info has been entered in the system for the given semester, the Student Info for that student has been entered in the system, the Students' Subject Choice Info has been entered in the system for that student in the given semester, and the Marks Info has been entered for that student for the given semester.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true appropriate error messages will be prompted to the user for doing the needful.

3.2.6 Report Generation

Student list reports

For each year a report will be generated containing the list of students enrolled in that batch year.

Report format:

Name of University
Name of the Program
List of students entrolled in year xxxx

S.No.	Student Enrollment Number	Student Name
1. 2.		

Students' subject choices list reports

For each batch year a report will be generated containing the list of students and their choices for Elective subjects in the selected semester. For Ist, IInd and IIIrd semesters the list will contain the names of elective subjects opted by each student. For IVth semester the list will contain the topic/subject area of dissertation for each student.

Report format (for Ist, IInd and IIIrd Semesters):

Name of University
Name of the Program

S.No.	Student Enrollment Number	Student Name	Elective 1		Elective 2	
			Code	Name	Code	Name
1. 2.						

Report format (for IVth Semester):

Name of University
Name of the Program

S.No.	Student Enrollment Number	Student Name	Topic/Subject Area of Dissertation
1. 2.			

Semester-wise mark lists

For each semester a mark list will be generated that will have the total marks (internal + external) of all students (enrolled in the selected Batch Year) of that semester in all subjects.

Report format:

Name of University
Name of the Program

S.No.	Enrollment Number	Subject 1 Total Marks	Subject 2 Total Marks	Subject 3 Total Marks	Subject 4 Total Marks	Subject 5 Total Marks	Subject 6 Total Marks
1. 2.							

Rank-wise list report

This report will be generated for each semester of every Batch Year. It will show the Grand Total marks, Percentage, Rank and Division secured of all students of that semester. The report will be sorted in increasing order of Percentage/Rank.

Report format:

Name of University
Name of the Program

S.No.	Enroll. No.	Name	Grand Total Marks	%age	Rank	Division
1. 2.						

3.2.7 User Accounts Information Maintenance

Description

The system will maintain information about various users who will be able to access the system. The following information would be maintained:

User Name, User ID, Password, and Role.

Validity checks

(i) Only user with role Administrator will be authorised to access the User Accounts Information Maintenance module.

(*ii*) User Name cannot be blank.
(*iii*) User ID cannot be blank.
(*iv*) User ID should be unique for every user.
(*v*) Password cannot be blank.
(*vi*) Role cannot be blank.

Sequencing information

User Account for a particular user has to be created in order for the system to be accessible to that user. At system startup, only a default user account for 'Administrator' would be present in the system.

Error handling/response to abnormal situations

If any of the above validations/sequencing flow does not hold true, appropriate error messages will be prompted to the user for doing the needful.

3.3 Performance Requirements

None

3.4 Design Constraints

None

3.5 Software System Attributes

3.5.1 Security

The application will be password protected. Users will have to enter correct username, password and role in order to access the application.

3.5.2 Maintainability

The application will be designed in a maintainable manner. It will be easy to incorporate new requirements in the individual modules (*i.e.*, subject info, student info, students' subject choices info, marks info, report generation and user accounts info).

3.5.3 Portability

The application will be easily portable on any windows-based system that has MS-Access 2000 installed.

3.6 Logical Database Requirements

The following information will be placed in a database:

(*i*) **Subject info:** Subject Name, Code, Credit points, Type and Semester
(*ii*) **Student info:** Student enrollment number, Student Name, enrollment year
(*iii*) **Students' subject choice info:** Student Enrollment No, Semester, Choice of Elective 1 subject, Choice of Elective 2 subject

(*iv*) **Marks info:** Student Enrollment No., Semester, internal Marks in each subject, External Marks in each subject
(*v*) **User account info:** User Name, User ID, Password, Role.

3.7 Other Requirements

None

REFERENCES

[BERR98] Berry D.M. & B. Lawrence, "*Requirements Engineering*", IEEE Software, Mar/April, 26–29, 1998.

[BITT03] Bittner K. and I spence, "*Use Case Modelling*", Pearson Education, 2003.

[BROO95] Brooks F.P., "No Silver Bullet", The Mythical Man Month: Essay on Software Engineering (Second Edition), Addison Wesley Longman, Reading, Mass, 179–209, 1995.

[CAIN75] Caine S. & E. Gordon, "*A Tool for Software Design*", Proceeding of the AFIPS, National Computer Conference, Vol. 47, AFIPS Press, Montvale, NJ, 271–276, 1975.

[CAPE94] Caper Jones, "*Charting the seas of Information Technology*", 1994.

[CARE90] Carey J. M., "*Prototyping: Alternative Systems Development Methodology*", Information and Software Technology, Vol. 32, No. 2, March 1990, 119–126.

[CERI88] Ceri S. et al., "*Software Prototyping by Relational Techniques: Expenses with Program Construction Systems*", IEEE Transaction on Software Engineering, Vol. 14, No. 11, Nov. 1988, 1597–1609.

[CHEN76] Chen P., "*The Entity–Relationship Model–Towards the Unified View of Data*", ACM Trans on database systems, March, 9–36, 1976.

[COAD89] Coad P. & E. Yourdon, "*OOA–Object Oriented Analysis*", Englewood Cliffs, N.J. Prentice Hall, 1989.

[COUN99] Councill B., "*Third–Party Testing and Stirrings of the New Software Engineering*", IEEE Software, Nov./Dec., 76–88, 1999.

[DAV194] Davis A. & P. Hsia, "*Giving Voice to Requirements Engineering*", IEEE Software, March, 12–16, 1994.

[DAVI90] Davis A.M., "*Software Requirements Analysis and Specification*", PH, Englewood Cliffs, NJ, 1990.

[DEMA79] DeMacro T.,"*Structured Analysis and System Specification*", Englewood Cliffs, N.J., PH, 1979.

[HEKM87] HeKmatpour S., "*Experience Evolutionary Prototyping in Large Software Project*", ACM Software Engineering Notes, Vol. 12, No. 1, January, 1987, 38–41.

[HEME82] Hemenway K. & L. X. McCusker, "*Prototyping & Evaluating a User Interface*", Proc. of the 6th International Computer Software & Applications Conference, Chicago, Illinois, Nov. 1982, 175–180.

[HICK03] Hickey A.M. and A.M. Davis, "*Requirements Elicitation and Elicitation Technique Selection: A Model for Two Knowledge Intensive Software Development Processes*",Proc. of 36th Hawai Int. conf. on system science, 2003.

[HOFF99] Hoffer J.A. et al., "*Modern Systems Analysis and Design*", Addison–Wesley, 1999.