Laboratory Manual

for

Software Engineering Laboratory

Subject Code – BCS 397

Credits -2 (0-0-3)

B.Tech. 6th Semester

Department of Computer Science and Engineering

Software Engineering Laboratory

AIM:

- To provide a deep insight into the importance of requirement modeling in the software industry.
- To improve their logical ability and creative skills for solving real world applications.

OBJECTIVES:

The objectives of the course are:

- To define, formulate and analyze a real world problem.
- To understand user conceptual models and thereby develop better specifications for them.
- Case studies are presented to demonstrate practical applications of different concepts.
- To provide a scope to students where they can solve small, real life problems either individually or in teams.
- To expose students on how to manage a project from beginning to end.

EVALUATION: (100 MARKS)

- Attendance – 10 Marks
- Assignment and Day-to-Day Performance – 20 Marks
- Laboratory Record – 20 Marks
- Laboratory Test – 30 Marks
- Viva-voce – 20 Marks

COURSE OUTCOME (CO):

- To understand requirement analysis and plan SRS based on their findings.
- To understand and design ER Diagram, DFD and Structure Chart Diagram.
- To study and design different UML diagrams.
- To design test cases based on the functional requirements.

CONTENTS

REFERENCES:

1. Mall Rajib, Fundamentals of Software Engineering, PHI.
2. Pressman Roger S., Software Engineering Practitioner's Approach, TMH.
3. Jalote Pankaj , An Integrated Approach to Software Engineering, Springer.
4. Sommerville Ian, Software Engineering, Pearson Education.

Assignment 1: Problem Statement and list of actors

Objective: To prepare the problem statement for any project and find the list of end users using the system.

Problem Description: The problem statement is the preliminary topic for any project. It is basically a one to three page statement describing the system form the client's point of view. Since it is from the client's point of view, it should be written using non-technical terms so as to serve a broad audience. The problem statement should not describe the solution for the problem. The end users (usually referred as actors) for the system are generally figured out from the problem statement. This problem statement would serve as an input to requirement engineering.

The following description serves as the problem statement for the case study.

Case Study: A Library Information System

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only. The final deliverable would a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg. passwords) is stored in plain text.

List of Actors:

- Students
- Employees
- Librarian

Conclusion: The problem statement was written successfully and list of actors were determined using the afore mentioned procedure.

Assignment 2: Requirement Analysis

Objective: To figure out the functional requirements, non-functional requirements, goals of implementation and constraints.

Problem Description: Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Somerville defines "requirement" as a specification of what should be implemented. Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems and customer dissatisfaction as well. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented. In this experiment we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification. After the completion of this experiment, students will be able to identify ambiguities, inconsistencies, incompleteness and the constraints imposed on the system from requirements specification.

Characteristics of Requirements: Requirements gathered for any new system to be developed should exhibit the following three properties:

Unambiguity: There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.

Consistency: To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that it the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.

Completeness: A particular requirement for a system should specify what the system should do and also what it should not. For example, consider software to be developed for ATM. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

Categorization of Requirements: Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

Functional requirements (FRs): These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.

Non-functional requirements (NFRs): They are not directly related functionalities that are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

Case Study: A Library Information System

Considering the LIS (as mentioned in Assignment 1), we need to identify some of the basic functionality of the system:

Identification of functional requirements

New user registration: Any member of the institute who wishes to avail the facilities of the library has to register himself with the Library Information System. On successful registration, a user ID and password would be provided to the member. He has to use these credentials for any future transaction in LIS.

Search book: Any member of LIS can avail this facility to check whether any particular book is present in the institute's library. A book could be searched by its: Title Authors name Publisher's name

User login: A registered user of LIS can login to the system by providing his employee ID and password as set by him while registering. After successful login, "Home" page for the user is shown from where he can access the different functionalities of LIS: search book, issue book, return book, reissue book. Any employee ID not registered with LIS cannot access the "Home" page -- a login failure message would be shown to him, and the login dialog would appear again. This same thing happens when any registered user types in his password wrong. However, if incorrect password has been provided for three time consecutively, the security question for the user (specified while registering) with an input box to answer it are also shown. If the user can answer the security question correctly, a new password would be sent to his email address. In case the user fails to answer the security question correctly, his LIS account would be blocked. He needs to contact with the administrator to make it active again.

Issue book: Any member of LIS can issue a book against his account provided that:

- The book is available in the library i.e. could be found by searching for it in LIS.
- No other member has currently issued the book.

- Current user has not issued the maximum number of books that he can.

If the above conditions are met, the book is issued to the member. Note that this FR would remain **incomplete** if the "maximum number of books that can be issued to a member" is not defined. We assume that this number has been set to five for each student and research scholars, and to ten for faculties. Once a book has been successfully issued, the user account is updated to reflect the same.

Return book: A book is issued for a finite time, which we assume to be a period of 15 days i.e., a book once issued should be returned within the next 15 days by the corresponding member of LIS. After successful return of a book, the user account is updated to reflect the same.

Reissue book: Any member who has issued a book might find that his requirement is not over within those 15 days. In that case, he might choose to reissue the book, and get permission to keep it for another 15 days. However, a member can reissue any book at most twice, after which he has to return it. Once a book has been successfully reissued, the user account is updated to reflect the information.

In a similar way we can list other functionality offered by the system as well. However, certain features might not be evident directly from the problem system, but which, nevertheless, are required. One such functionality is "User Verification". The LIS should be able to judge between a registered and non-registered member. Most of the functionality would be available to a registered member. The "New User Registration" would, however, be available to non-members. Moreover, an already registered user shouldn't be allowed to register himself once again. Having identified the (major) functional requirements, we assign an identifier to each of them for future reference and verification.

Requirement	Priority
R1 New user registration	High
R2 User Login	High
R3 Search book	High
R4 Issue book	High
R5 Return book	High
R6 Reissue book	Low

The above table shows the priority assigned to each requirement.

Identification of non-functional requirements Having discussed about functional requirements, let's try to identify a few non-functional requirements.

Performance Requirements:

- This system should remain accessible 24x7.
- At least 50 users should be able to access the system altogether at any given time.

Security Requirements:

- This system should be accessible only within the institute LAN.
- The database of LIS should not store any password in plain text -- a hashed value has to be stored.

Software Quality Attributes

Database Requirements

Design Constraints:

- The LIS has to be developed as a web application, which should work with Firefox 5, Internet Explorer 8, Google Chrome 12, Opera 10
- The system should be developed using HTML 5

Conclusion: The functional requirements, non-functional requirements, goals of implementation and constraints were discovered and written successfully.

Assignment 3: Software Requirement Specification (SRS) Document

Objective: Understanding and constructing a SRS

Problem Description: A SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time usually prior to any actual design or development work. It's a two way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time. The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements we need to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

The SRS may be one of a contract deliverable Data Item Descriptions or have other forms of organizationally-mandated content.

Characteristics of a Good Software Requirements Specification:

A software requirements specification should be clear, concise, consistent and unambiguous. It must correctly specify all of the software requirements. However the software requirement specification should not describe any of the design or verification aspects, except where constrained by any of the stake holder's requirements.

- **Complete**

  For a software requirements specification to be complete, it must have the following properties:

  - Description of all major requirements relating to functionality, performance, design constraints and external interfaces.
  - Definition of the response of the software system to all reasonable situations.
  - Conformity to any software standards, detailing any sections which are not appropriate.
  - Have full labelling and references of all tables and references, definitions of all terms and units of measure.

- **Consistent**

  A software requirement specification is consistent if none of the requirements conflict.

  There are a number of different types of confliction:

- Multiple descriptors - This is where two or more words are used to reference the same item, i.e. where the term cue and prompt are used interchangeably.
- Opposing physical requirements - This is where the description of real world objects clash, e.g. one requirement states that the warning indicator is orange, and another states that the indicator is red.
- Opposing functional requirements - This is where functional characteristics conflict, e.g. perform function X after both A and B has occurred, or perform function X after A or B has occurred.

- **Traceable**

  A software requirement specification is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand who asked for the requirement and also what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references is used to aid the modification of future documents by stating where a requirement has been referenced.

- **Unambiguous**

  As the Oxford English dictionary states the word unambiguous means [Hawkins ’88]: "not having two or more possible meanings". This means that each requirement can have one and only one interpretation. If it is unavoidable to use an ambiguous term in the requirements specification, then there should be clarification text describing the context of the term.

- **Verifiable**

  A software requirement specification is verifiable if all of the requirements contained within the specification are verifiable. A requirement is verifiable if there exists a finite cost-effective method by which a person or machine can check that the software product meets the requirement. Non-verifiable requirements include "The system should have a good user interface" or "the software must work well under most conditions" because the performance words of good, well and most are subjective and open to interpretation. If a method cannot be devised to determine whether the software meets a requirement, then the requirement should be removed or revised.

Conclusion: The Software Requirement Specification document was prepared as per the given IEEE SRS Template.

SAMPLE IEEE SRS TEMPLATE

Software Requirements Specification

for

<Project>

Version 1.0 approved

Prepared by <author>

<organization>

<date created>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

1.2 Document Conventions

<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>

1.3 Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

1.4 Product Scope

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Overall Description

2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and

identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2 Product Functions

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>

2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>

2.4 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.6 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

2.7 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend

to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

3. External Interface Requirements

3.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

3.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

3.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

3.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

4. System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

4.1 System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

4.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

4.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

4.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:
REQ-2:

4.2 System Feature 2 (and so on)

5. Other Nonfunctional Requirements

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>

Assignment 4: Entity Relationship Model

Objective: Understanding the Entity Relation Model from the Problem Statement

Problem Description: Creating databases is a very important task for a system to function properly. Before going to form the exact database tables and establishing relationships between them, we conceptually or logically can model our database using ER diagrams. In this experiment we will learn how to find the entity sets, their attributes and how the relationships between the entities can be established for a system.

- **Entity-Relationship model:** It is used to represent a logical design of a database to be created. In ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modelled as relationships. For example, student and school -- they are two entities. Students study in school. So, these two entities are associated with a relationship "Studies in".
- **Entity Set and Relationship Set**: An entity set is a collection of all similar entities. For example, "Student" is an entity set that abstracts all students. Similarly, a "Relationship" set is a set of similar relationships.
- **Attributes of Entity**: Attributes are the characteristics describing any entity belonging to an entity set. Any entity in a set can be described by zero or more attributes. For example, any student has got a name, age, an address. At any given time a student can study only at one school. In the school he would have a roll number, and of course a grade in which he studies. These data are the attributes of the entity set Student.
- **Keys** One or more attribute(s) of an entity set can be used to define the following keys:
- **Super key:** One or more attributes, which when taken together, helps to uniquely identify an entity in an entity set. For example, a school can have any number of students. However, if we know grade and roll number, then we can uniquely identify a student in that school.
- **Candidate key:** It is a minimal subset of a super key. In other words, a super key might contain extraneous attributes, which do not help in identifying an object uniquely. When such attributes are removed, the key formed so is called a candidate key.
- **Primary key:** A database might have more than one candidate key. Any candidate key chosen for a particular implementation of the database is called a primary key. Prime attribute: Any attribute taking part in a super key.
- **Weak Entity** An entity set is said to be weak if it is dependent upon another entity set. A weak entity can't be uniquely identified only by it's attributes. In other words, it doesn't have a super key. For example, consider a company that allows employees to have travel allowance for their immediate family. So, here we have two entity sets: employee and family, related by "Can claim for". However, family doesn't have a super key. Existence of a family is entirely dependent on the concerned employee. So, it is meaningful only with reference to employee.

- **Entity Generalization and Specialization:** Once we have identified the entity sets, we might find some similarities among them. For example, multiple people interact with a banking system. Most of them are customers, and rest employees or other service providers. Here, customers, employees are persons, but with certain specializations. Or in other way, person is the generalized form of customer and employee entity sets. ER model uses the "IS A" hierarchy to depict specialization (and thus, generalization).

- **Mapping Cardinalities**

  One of the main tasks of ER modeling is to associate different entity sets.

  Let's consider two entity sets E1 and E2 associated by a relationship set R. Based on the number of entities in E1 and E2 are associated with, we can have the following four type of mappings:

  One to one: An entity in E1 is related to at most a single entity in E2, and vice versa .

  One to many: An entity in E1 could be related to zero or more entities in E2. Any entity in E2 could be related to at most a single entity in E1.

  Many to one: Zero or more number of entities in E1 could be associated to a single entity in E2. However, an entity in E2 could be related to at most one entity in E1.

  Many to many: Any number of entities could be related to any number of entities in E2, including zero, and vice versa.

Given a problem statement, the first step is to identify the entities, attributes and relationships. We represent them using an ER diagram. Using this ER diagram, table structures are created, along with required constraints. Finally, these tables are normalized in order to remove redundancy and maintain data integrity. Thus, to have data stored efficiently, the ER diagram is to be drawn as much detailed and accurate as possible.

Case Study: A Library Information System (LIS)

Considering the LIS (as mentioned in Assignment 1), we need to identify some of the basic functionality of the system:

The first step towards ER modeling is to identify the set of relevant entities from the given problem statement. The two primary, and obvious, entity sets in this context are "Member" and "Book". The entity set "Member" represents all students, professors, or employees who have registered themselves with the LIS. While registering with the LIS one has to furnish a lot of personal and professional information. This typically includes name, employee ID (roll for students), email address, phone, age, date of joining in the institute. The system may store some not-so-important information as well like, blood group, marital status, and so on. All these pieces

of information that a user has to provide are sufficient to describe a particular member. These characteristics are the attributes of the entities belonging to the entity set "Member". It is essential for an entity to have one or more attributes that help us to distinguish it from another entity. 'Name' can't help that -- two persons could have exactly the same name. However, ('Name', 'Phone') combination seems to be okay. No two persons can have the same phone number. 'Employee ID', 'Email address' are other potential candidates. Here, 'Employee ID', 'Email address' and ('Name', 'Phone') are super keys. We choose 'Employee ID' to uniquely identify a user in our implementation. So, 'Employee ID' becomes our primary key (PK) for the "Member" entity set. Figure 1 represents this set along with it's attributes and the primary key.

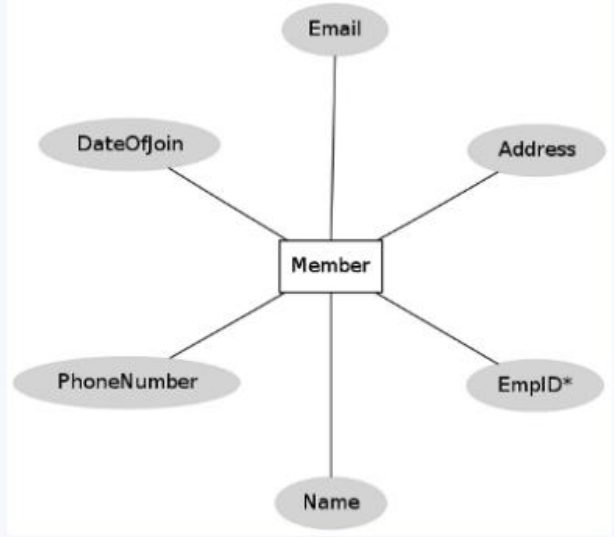


Figure 1: "Member" entity set

Let us now focus on the "Book" entity set. Typical attributes of a book are it's title, name of author(s), publisher, date of publication, edition, language, ISBN-10, ISBN-13, price, date of purchase. The set of listed attributes for a book doesn't give a straight forward choice of primary key. For instance, several books could have the same title. Again, ISBN numbers for a book are specific to it's edition - it can't distinguish between two books of the same edition. One might be tempted to use a combination of ('Title', 'Authors') as a primary key. This has some shortcomings. It is advisable not to use texts as a PK. Moreover, the number of authors that a book could have is not fixed, although it is a small, finite number. The rules of normalization (not covered here) would dictate to have a separate field for each author like 'Author1', 'Author2', and so on. Therefore, we assign an extra attribute, 'ID', to each book as it's PK. Different databases available in the market provide mechanisms to generate such an unique ID, and automatically increment it whenever a new entity is added. A graphical representation of the "Book" entity set is shown in figure 2.

Figure 2: "Book" entity set

One point to note here is that a book is likely to have multiple copies in the library. Therefore, one might wish to have a 'no. of copies' attribute for the "Book" entity set. However, that won't allow us to differentiate among the different copies of book bearing same title by same author(s), edition, and publisher. The approach that we have taken is to uniquely identify each book even though they are copies of the same title. To buy any new book an order is to be placed to the distributor. This task is done by the librarian. Therefore, "Librarian" and "Distributor" are two other entities playing roles in this system. Having identified the key entities, we could now relate them with each other. Let us consider the entity sets "Member" and "Book". A member can issue books. In fact, he can issue multiple books up to a finite number say, N. A particular book, however, could be issued by a single member only. Therefore, we have a one-to-many mapping from "Member" to "Book" entity sets. This relationship between "Member" and "Book" entity sets is pictorially depicted in figure 3.

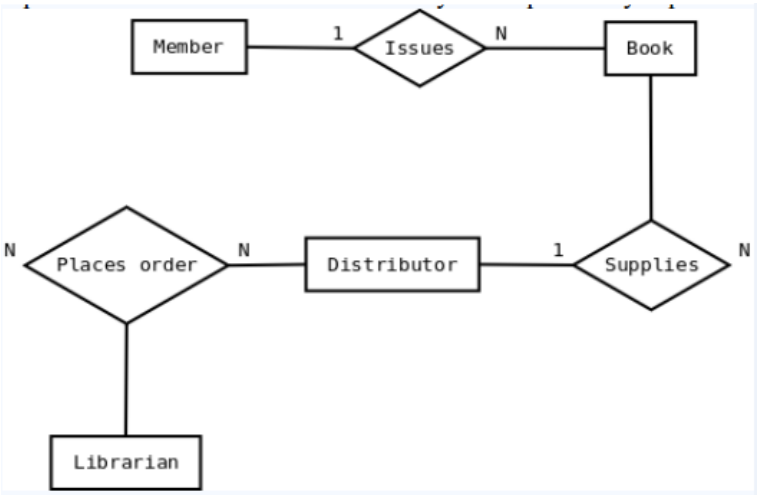


Figure 3: Relationships among different entity sets

Figure 3 also shows that the librarian can "place order" for books to the distributor. This is a many-to-many mapping since a librarian can purchase books from multiple distributors. Also, if the institute has more than one librarians (or any other staff having such authority), then each of them could place order to the same distributor. An order is termed as complete when distributor supplies the book(s) and invoice.

Therefore, the baseline ER model so prepared should be revised by considering the business model yet again to ensure that all necessary information has been captured. Once this has been finalized, the next logical step would be to create table structures for each identified entity set (and relationships in some cases) and normalize the relations

Conclusion: The Entity Relationship Diagram was designed successfully using the above steps.

Assignment 5: Data Flow Diagram and Structure Chart Diagram

Objective: Understanding and preparing the Data Flow Diagram and Structure Chart Diagram

Problem Description: Data Flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. As the name indicates, its focus is on the flow of information, where data comes from, where it goes and how it gets stored. The DFD does not mention anything about how data flows through the system.

DFD Components: DFDs can represent source, destination, storage and flow of data using the following set of components -

- **Entities** - Entities are source and destination of information data. Entities are represented by rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by circle or round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

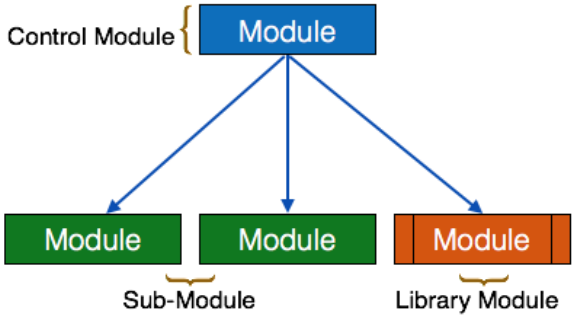
Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.
- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.
- Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

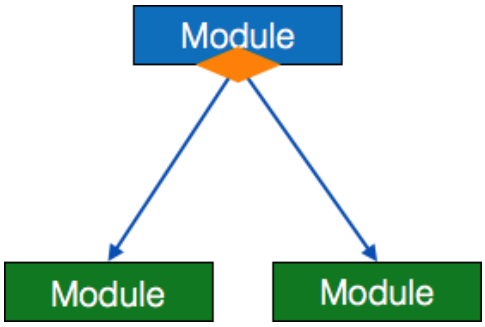
Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD. Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

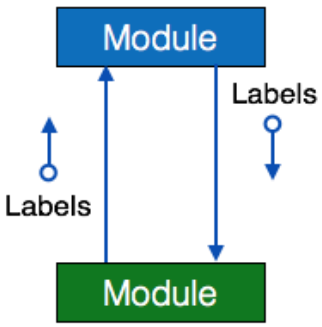
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



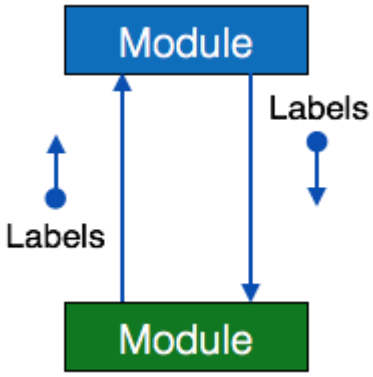
- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



- **Data flow** - A directed arrow with empty circle at the end represents data flow.



- **Control flow** - A directed arrow with filled circle at the end represents control flow.



Conclusion: The Data Flow Diagram and Structure Chart were designed successfully using the above notations.

Assignment 6: Use Case Diagram and Use Case Specification Document

Objective: To identify use cases and thereby create use case diagram and the Use Case Specification Document.

Problem Description: Use Case Diagrams are used to describe a set of actions (use cases) that some system or systems should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Guidelines for use case diagram:

1. **Use Case**- A use case describes a sequence of actions that provides measureable value to an actor. A use case is represented by a horizontal ellipse.

    - **Names begin with a verb** – Since a use case models an action so the name should begin with a verb.
    - **Make the name descriptive** – This is to give more information for others who are looking at the diagram. For example "Print Invoice" is better than "Print".
    - **Highlight the logical order** – For example if you're analyzing a bank customer typical use cases include open account, deposit and withdraw. Showing them in the logical order makes more sense.
    - **Place included use cases to the right of the invoking use case** – This is done to improve readability and add clarity.
    - **Place inheriting use case below parent use case** – Again this is done to improve the readability of the diagram.

2. **Actors-** An actor is a person, organization or external system that plays a role in one or more interactions with your system. They are usually represented as a stick figure.

    - **Give meaningful business relevant names for actors** – For example if your use case interacts with an outside organization its much better to name it with the function rather than the organization name. (Eg: Airline Company is better than PanAir)**.**
    - **Primary actors should be to the left side of the diagram** – This enables you to quickly highlight the important roles in the system.
    - **Actors model roles (not positions)** – In a hotel both the front office executive and shift manager can make reservations. So something like "Reservation Agent" should be used for actor name to highlight the role.

- **External systems are actors** – If your use case is send email and if interacts with the email management software then the software is an actor to that particular use case.
- **Actors don't interact with other actors** – In case actors interact within a system you need to create a new use case diagram with the system in the previous diagram represented as an actor.
- **Place inheriting actors below the parent actor** – This is to make it more readable and to quickly highlight the use cases specific for that actor.

3. **Relationships**

- Arrow points to the base use case when using <<extend>>.
- <<extend>> can have optional extension conditions.
- Arrow points to the included use case when using <<include>>.
- Both <<extend>> and <<include>> are shown as dashed arrows.
- Actor and use case relationship doesn't show arrows.

4. **Systems / Packages**

- Use them sparingly and only when necessary.
- Give meaningful and descriptive names to these objects

With the above mentioned guidelines, the corresponding use case diagram for a case study can be drawn. Along with the diagram, the students are supposed to produce a use case specification document.

USE CASE SPECIFICATION DOCUMENT

Use Case Specification : <Use-Case Name>

1. Use Case Name

1.1 Brief Description

[The description should briefly convey the role and purpose of the use case. A single paragraph should suffice for this description.]

2. Flow of Events

2.1 Basic Flow

[This use case starts when the actor does something. An actor always initiates use cases. The use case should describe what the actor does and what the system does in response. It should be phrased in the form of a dialog between the actor and the system. The use case should describe what happens inside the system, but not how or why. If information is exchanged be specific about what is passed back and forth. For example, it is not very illuminating to say that the Actor enters customer information; it is better to say the Actor enters the customer's name and address.]

2.2 Alternative Flows

2.2.1 < First Alternative Flow>

[More complex alternatives should be described in a separate section, which is referred to in the basic flow of events section. Think of the alternative flow sections like alternative behavior – each alternative flow represents alternative behavior (many times, because of exceptions that occur in the main flow). They may be as long as necessary to describe the events associated with the alternative behavior. When an alternative flow ends, the events of the main flow of events are resumed unless otherwise stated.]

2.2.1.1 <An Alternative sub-flow>

[Alternative flows may in turn be broken down into sub-sections if it improves clarity.]

2.2.2 <Second Alternative Flow>

[There may be, and most likely will be, a number of alternative flows in a use case. Keep each alternative separate to improve clarity. Using alternative flows improves the readability of the use case, as well as preventing use cases from being decomposed into hierarchies of use cases. Keep in mind that use cases are just textual descriptions, and their main purpose is to document the behavior of a system in a clear, concise and understandable way.]

3. Special Requirements

[A Special Requirement is typically a non-functional requirement that is specific to a use case but is not easily or naturally specified in the text of the use case's event flow. Examples of special requirements include legal and regulatory requirements, application standards, and quality attributes of the system to be

built, including usability, reliability, performance or supportability requirements. Additionally, other requirements such as operating systems and environments, compatibility requirements, and design constraints should be captured in this section.]

3.1 <First special requirement>

4. Pre Conditions

[A precondition (of a use case) is the state of the system that must be present prior to a use case being performed.]

4.1 <Precondition One>

5. Post Conditions

[A post condition (of a use case) is a list of possible states the system can be in immediately after a use case has finished.]

5.1 <Post condition one>

Conclusion: The use cases were identified and the use case diagram was prepared likewise. The use case specification template was also documented.

Assignment 7: Class Diagram

Objective: To draw a class diagram.

Problem Description: The class diagram is a static diagram. It represents the static view of an application. The class diagram describes the attributes and operations of a class and also the constraints imposed on the system.

Guidelines for class diagram:

The following points should be remembered while drawing a class diagram:

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Put common terminology for names.
- Choose complete singular nouns over class names.
- Name the operations with a strong verb.
- Name the attributes with a domain-based noun.
- Responsibility (attributes and methods) of each class should be clearly identified.
- List static operations/attributes before instance operations/attributes.
- List operations/attributes in decreasing visibility.
- Develop consistent method signatures.
- Each element and their relationships should be identified in advance.
- For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.

How to draw a class diagram

When designing classes, the attributes and operations it will have are observed first. Then determine how many instances of the classes will interact with each other. Relate the classes with appropriate associations. Using just these basic techniques, one can develop a complete view of the software system.

Conclusion: The class diagram was successfully drawn using the afore mentioned procedure.

Assignment 8: Sequence Diagram

Objective: To design the sequence diagram.

Problem Description: Sequence diagrams assist the detailing and specification of business use cases by emphasizing message exchange. The various scenarios of a business use case can be depicted in a sequence diagram. The representation is restricted to the message exchange within each business use case. Generally, the level of detail for these sequence diagrams is higher than for sequence diagrams spanning use cases.

Construction of Sequence Diagram

- Designate Actors and Business System—Who is Taking Part?

  Sequence diagrams illustrate the interactions between actors and the business system. Fundamentally we have a pool of interaction partners from the use case diagrams. Depending on the flow that is being depicted in the sequence diagram, the appropriate actors and business systems can be selected from this pool.

- Designate Initiators—Who Starts the Interactions?

  For every sequence of interactions the actor who starts the interaction has to be identified. This actor is called the *initiator*. Since in the external view of the business model each business use case is initiated by an actor, we can here also select the actor from the pool of actors in the use case diagrams.

- Describe the Message Exchange between Actors and the Business System—Which Messages are being exchanged?

  After the initiator has been defined, the subsequent progression of interactions has to be identified. For each communication step it has to be determined what information is exchanged. In this way the message will be defined. Messages are requests to do something directed toward a particular partner. The business objects that are exchanged with these messages also have to be defined.

- Identify the Course of Interactions—What is the Order?

  All messages are exchanged in a chronological order that has to be identified. Messages are inserted along the *y*-axis in increasing chronological order, from top to bottom.

- Insert Additional Information—What Else is Important?

  Important activities of involved actors and business systems and important conditions can be inserted into the diagram as comments. Comments are inserted at the level of the appropriate message. Restrict this to important comments that have significance so that the diagram is not overcrowded with text.

The sequence diagrams are designed with respect to the actors. The course of events that an actor invokes can either constitute the primary flow or the alternate flow. This is very well evident from the use case diagram. Based on this, each actor will have primary/main sequence diagram and alternate sequence diagrams.

Conclusion: The primary and alternate sequence diagram was designed successfully for every actor of the system.

Assignment 9: Activity Diagram, State Chart Diagram and Collaboration Diagram

Objective: To design the Activity Diagram, State Chart Diagram and Collaboration Diagram

Problem Description: Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

General Guidelines

Place the start point in the Top-Left Corner. A start point is modeled with a filled in circle. Every UML Activity Diagram should have a starting point.

Always include an ending point. An ending point is modeled with a filled in circle with a border around it

Activities- An activity, also known as an activity state, on a UML Activity diagram typically represents the invocation of an operation, a step in a business process, or an entire business process. While representing activities, the following two points have to be focused on at each stage.

- Question "Black Hole" Activities. A black hole activity is one that has transitions into it but none out, typically indicating that you have either missed one or more transitions.
- Question "Miracle" Activities. A miracle activity is one that has transitions out of it but none into it, something that should be true only of start points.

Decision Points- A decision point is modeled as a diamond on a UML Activity diagram. While representing decision points, the following two points have to be focused on at each stage.

- Decision Points Should Reflect the Previous Activity.
- Avoid Superfluous Decision Points.

Parallel Activities- Parallel activities are depicted using two parallel bars. The first bar is called a fork; it has one transition entering it and two or more transitions leaving it. The second bar is a join, with two or more transitions entering it and only one leaving it. While representing parallel activities, the following points have to be focused on at each stage.

- A Fork Should Have a Corresponding Join.
- Forks Have One Entry Transition.
- Joins Have One Exit Transition

Swim lane - A swim lane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread. While representing swim lanes, the following points have to be focused on at each stage.

- Order swim lanes in a logical manner.
- Apply swim lanes to linear processes.
- Have less than five swim lanes.
- Consider swim areas for complex diagrams.
- Consider horizontal swim lanes for business processes.

State Chart Diagram

A State chart diagram describes the different states of a component in a system. States are defined as a condition in which an object exists. The states are specific to a component/object of a system. It describes the flow of control from one state to another state. These states are changed when some event is triggered. So state chart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

General Guidelines

State chart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Before drawing a State chart diagram we need to clarify the following points:

- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

Identifying these three parameters will help us to draw the corresponding state chart.

Collaboration Diagram

Collaboration diagrams are relatively easy to draw. They are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaborations are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces. The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects.

Conclusion: The Activity Diagram, State Chart Diagram and Collaboration Diagram were designed successfully following the above mentioned guidelines.

Assignment 10: Test Coverage Metrics

Objective: To design test cases based on requirements and design

Problem Description: Development of new software, like any other product, remains incomplete until it subjected to exhaustive tests. The primary objective of testing is not only to verify that all desired features have been implemented correctly, but also includes the verification of the software behavior in case of "bad inputs". In this experiment we discuss how to design test cases for the functionalities.

Testing is the process of executing a program with the intent of finding errors. A good test case is one with a high probability of finding an as-yet undiscovered error. A successful test is one that discovers an as-yet-undiscovered error. The causes of the software defects are: specification may be wrong; specification may be a physical impossibility; faulty program design; or the program may be incorrect.

A test case is a document, which has a set of test data, preconditions, expected results and post conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values; the application has a definitive outcome and leaves the system at some end point or also known as execution post condition.

Typical Test Case Parameters:

- Test Case ID
- Test Scenario
- Test Case Description
- Test Steps
- Prerequisite
- Test Data
- Expected Result
- Test Parameters
- Actual Result
- Environment Information
- Comments

Test suite is a collection of set of tests which helps testers in executing and reporting the test execution status. It can take any of the three states namely Active, In progress and completed. Test suites are created based on the cycle or based on the scope. It can contain any type of tests, viz - functional or Non-Functional.

General Guidelines for writing effective test cases

- First understand the requirements & while *writing test cases* do not assume any requirements by your own. Raise the question which is not clear in requirement or requirements are misleading or incomplete, feel free to ask questions to your business analyst or client. Don't ask to developers on this.
- Prior to design the test cases figure out all features of application.
- Ensure that test case should cover all functionality mention in requirement document.
- Avoid repetition of test cases which help to get exact scope of testing.
- Generic test cases should be collected & combine together in test suite, which helps to minimize the effort of writing standard common test cases every time and can be used over the project life cycle.
- Testing Priority should be assign to each test case. Select the Test case priority depending upon how important the test case is for that Feature, component or the product. In execution queue, high priority test cases should be picked first then medium and then Low priority test cases.
- As end users or clients are always interested in reports, so test cases should be group properly (Phase I, II wise, Module wise, Sprint wise or User story wise if *Agile methodology*), so end user will come to know about Quality of the product based on test case execution (number of test cases Pass/Failed).
- While writing test cases keep in mind all your **test cases should be simple and easy to understand**. Don't write explanations like essays. Be to the point.
- Keep in mind input data for test cases is very important part in testing; your test cases should validate range of input data. Also check how system behaves in the normal & abnormal conditions.
- While **writing test cases** you should concentrate on real life scenarios first which end user going to use day to day life activity and accordingly test cases should be prepare.

- Every test case may or may not have defect linked but each defect should have test case linked.

Conclusion: Following the above guidelines, test cases for a system were designed successfully.

LIST OF CASE STUDIES

The experiments are supposed to be carried out on the following case studies

1. **Name of the Project**: *Patient Billing Software*

   Objective/Vision: This project is aimed at developing a patient billing software system that is of importance to a hospital. The PBS is a local software system. This system can be used to maintain the location (bed no.) of each patient either in the ward or the ICU. Information about the patient and the charges to be paid is also stored

2. **Name of the Project:** *Back To My Village*

   Objective/Vision: BTMV is a charity group of professionals those want to voluntarily contribute in their village/town's development. Issues like Primary education, people's health, government policies awareness and availability of basic facilities/infrastructure are on main focus among others. Through the website group want to help their members collaborate, to plan, assess and implement different activities and learn with others experience/feedbacks/suggestions. Group also wants to encourage others to join their initiatives and recognize their contributions.

3. **Name of the Project**: *Chess Masters Club*

   Objective/Vision: Creating and managing an Online Chess Club where game lovers can learn and play Chess games by different means (Chess tutorial, puzzle, game with computer or other player etc.). Professional players can take part in tournaments that is totally a commercial activity for site owner.

4. **Name of the Project**: *Constituency Management System*

   Objective/Vision: Looking for a comprehensive solution to e-governance so that various constituencies can be managed and a complete solution is obtained. This will have all constituencies managed by respective MLA/MP and each of the funds used/left will be shown. The residents can also complain over any problem they are facing and will have their queries responded.

5. **Name of the Project:** *Customer Service Desk*

   Objective/Vision: An online comprehensive customer care solution to manage customer interaction and complaints with the service providers by phone, mobile, web and e-mail. The system should have capability to integrate with any service provider from any domain or industry like banking, telecom, railways etc.

6. **Name of the Project**: *E-Mentoring for Women*

   Objective/Vision: Developing an online mentoring system to promote more women to splurge into the field of Science and technology breaking the myths and taboo's society imposes. Also, to give them a platform to be on power with a working woman

7. **Name of the Project:** *Get Your Campus*

   Objective/Vision: A state-level online comprehensive solution to provide information regarding various institutes, their courses and admission procedures for admission seekers in Bachelor courses. Besides this, live/offline counseling to students for carrier prospects by experts.

8. **Name of the Project**: *Go Vegetarian*

   Objective/Vision: This is a social group of Indian NRIs those are committed to promote the benefits of vegetarian foods all over the world and also want to expand group. Through the website they want to impart knowledge on vegetarian food preparation, impact on health, information about meetings, events and product promotions etc. Group members are also convincing stores, food chain giants to prepare and market certain vegetarian dishes so all group members' wants to update their activities.

9. **Name of the Project**: *GSM based Remote Monitoring and Billing*

   Objective/Vision: The vision of the project is to replace the existing manual reading of electricity meters installed throughout the country (Home, Agricultural, and Industrial). The proposed solution is to build a server for the Electricity boards in each state where the custom built GSM meters would update in real time through SMS and instant status of the meter network can be established. The system will cut costs and improve transparency to a very large extent. Any failure or inconvenience on the consumer side can be instantly detected and rectified. The electricity board server can monitor and analyze the status of each and every individual meter on the network. The server would also provide a complete billing solution for the same.

10. **Name of the Project:** *Improved Railway Ticketing*

    Objective/Vision: Looking for an online solution to provide travelers with all facilities on internet. It will provide the following facilities on one site:
    * Information regarding trains
    * Seat Availability
    * Online Reservation/ Cancellation
    * Info about Train Running

11. **Name of the Project**: *Insurance on Internet*

    Objective/Vision: Objective is to automate all possible functionalities of the insurance sector by helping all stakeholders to use this web-based system. Obviously, company is looking for more profit, bigger market reach and 24X7 availability of services.

12. **Name of the Project** *Internet banking system*

    Objective/Vision Looking for an online comprehensive solution to manage Internet banking. This will be accessible to all customers who have a valid User Id and Password. This system provides the following facilities: Balance Enquiry, Funds Transfer to another account in the same bank, Request for cheque book/change of address/stop payment of cheques, viewing Monthly and annual statements.

13. **Name of the Project:** *IT Enabled Academia*

    Objective/Vision: Your solution is supposed to automate the routine teaching-learning support activities so the academics authority can monitor it and control the things. It starts from Time-table generation on available resources like programmes, courses, subjects, teachers, lecture room/lab preferences. Assignments/lecture notes upload; Daily class attendance, student feedback, exams marks updation and students/faculty profile management would be the integral part of this system.

14. **Name of the Project:** *Manage Prisons*

    Objective/Vision: This project is aimed at developing a prison management system that is a collection of registers and reports for the effective management of prisons. Besides this police and government officials can see crime/criminals reports for their purpose.

15. **Name of the Project:** *My Mission - City Without Crime*

    Objective/Vision: Developing an online comprehensive crime reporting system to engage public, NGOs, police and government agencies to be more quick, proactive and responsive to fight with crime and criminals.

16. **Name of the Project**: *News That Matters*

    Objective/Vision: A national newspaper agency wants to automate its customer facing services online (besides hard-copy version) and also want to facilitate a collaborative environment to geographically dispersed teams to make them more innovative and ahead of time.