System Requirements Specification

for

<Project Name>

Version 1.0

Prepared by

<author name, registration no.>

<author name, registration no.>

<author name, registration no.>

Group Code

**<Group Code> − SA1**

B.Tech. CSE, Section – E,

LAB. Batch – 1

Campus, ABC University

<dd/mmm/yyyy>

Table of Contents

[1 Introduction 4](#_Toc298883550)

[1.1 Purpose 4](#_Toc298883551)

[1.2 Scope 4](#_Toc298883552)

[1.3 Definitions, Acronyms, and Abbreviations 4](#_Toc298883553)

[1.4 References 4](#_Toc298883554)

[1.5 Overview 4](#_Toc298883555)

[2 Overall Description 5](#_Toc298883556)

[2.1 Product Perspective 5](#_Toc298883557)

[2.2 Product Functions 5](#_Toc298883558)

[2.3 User Characteristics 5](#_Toc298883559)

[2.4 General Constraints 5](#_Toc298883560)

[2.5 Assumptions and Dependencies 5](#_Toc298883561)

[3 Specific Requirements 6](#_Toc298883562)

[3.1 External Interface Requirements 6](#_Toc298883563)

[3.1.1 User Interfaces 6](#_Toc298883564)

[3.1.2 Hardware Interfaces 6](#_Toc298883565)

[3.1.3 Software Interfaces 6](#_Toc298883566)

[3.1.4 Communication Interfaces 6](#_Toc298883567)

[3.2 Functional Requirements 6](#_Toc298883568)

[3.2.1 Mode 1 7](#_Toc298883569)

[3.2.2 Mode *m* 7](#_Toc298883570)

[3.3 Performance Requirements 7](#_Toc298883571)

[3.4 Design constraints 7](#_Toc298883572)

[3.5 Attributes 7](#_Toc298883573)

[3.6 Other Requirements 8](#_Toc298883574)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The COVID-19 pandemic has posed significant challenges to blood banks and blood donations, including disruptions in donor engagement, collection logistics, and ensuring the safety of donors and staff. Resolving these issues is essential to sustain an effective blood supply chain in the face of ongoing public health challenges.

## 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. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.

FRONT END :HTML, CSS, JAVA SCRIPT

Back end:

DATA BASE:ORACLE, SQL.

REQUIRMENTS:ROUTER, DATABASE MANAGEMENT.

HARDWARE:COMPUTERS,

## Definitions, Acronyms, and Abbreviations

CASE tools known as Computer-aided software engineering tools is a kind of component-based development which allows its users to rapidly develop information systems. The main goal of case technology is the automation of the entire information systems development life cycle process using a set of integrated software tools, such as modeling, methodology and automatic code generation. Component based manufacturing has several advantages over custom development. The main advantages are the availability of high quality, defect free products at low cost and at a faster time. The prefabricated components are customized as per the requirements of the customers. The components used are pre-built, ready-tested and add value and differentiation by rapid customization to the targeted customers. However the products we get from case tools are only a skeleton of the final product required and a lot of programming must be done by hand to get a fully finished, good product.

# CHARACTERISTICS OF CASE:

Some of the characteristics of case tools that make it better than customized development are;

* It is a graphic oriented tool.
* It supports decomposition of process.

Some typical CASE tools are:

* Unified Modeling Language
* Data modeling tools, and
* Source code generation tools

# INTRODUCTION TO UML (UNIFIED MODELING LANGUAGE):

The UML is a language for specifying, constructing, visualizing, and documenting the software system and its components. The UML is a graphical language with sets of rules and semantics. The rules and semantics of a model are expressed in English in a form known as OCL (Object Constraint Language). OCL uses simple logic for specifying the properties

of a system. The UML is not intended to be a visual programming language. However it has a much closer mapping to object-oriented programming

languages, so that the best of both can be obtained. The UML is much simpler than other methods preceding it. UML is appropriate for modeling systems, ranging from enterprise information system to distributed web based application and even to real time embedded system. It is a very expensive language addressing all views needed to develop and then to display system even though understand to use. Learning to apply UML effectively starts forming a conceptual mode of languages which requires learning.

Three major language elements:

* UML basic building blocks
* Rules that dictate how this building blocks put together
* Some common mechanism that apply throughout the language The primary goals in the design of UML are:
  + 1. Provides users ready to use, expressive visual modeling language as well so they can develop and exchange meaningful models.
    2. Provide extensibility and specialization mechanisms to extend the core concepts.
    3. Be independent of particular programming languages and development processes.
    4. Provide formal basis for understanding the modeling language.
    5. Encourage the growth of the OO tools market.
    6. Support higher-level development concepts.
    7. Integrate best practices and methodologies.

Every complex system is best approached through a small set of nearly independent views of a model. Every model can be expressed at different levels of fidelity. The best models are connected to reality. The UML defines nine graphical diagrams:

1. Class diagram
2. Use-case diagram
3. Behavior diagram
   1. Interaction diagram
      1. sequence diagram
      2. collaboration diagram
   2. state chart diagram
   3. activity diagram
4. Implementation diagram

4.1component diagram 4.2deployment diagram

# UML class diagram:

The UML class diagram is also known as object modeling. It is a static analysis diagram. These diagrams show the static structure of the model. A class diagram is a connection of static model elements, such as classes and their relationships, connected as a graph to each other and to their contents.

# Use-case diagram:

The functionality of a system can be described in a number of different use-cases, each of which represents a specific flow of events in a system. It is a graph of actors, a set of use-cases enclosed in a boundary, communication, associations between the actors and the use-cases, and generalization among the use-cases.

# Behavior diagram:

It is a dynamic model unlike all the others mentioned before. The objects of an object oriented system are not static and are not easily understood by static diagrams. The behavior of the class’s instance (an object) is represented in this diagram. Every use-case of the system has an associated behavior diagram that indicates the behavior of the object. In conjunction with the use-case diagram we may provide a script or interaction diagram to show a time line of events. It consists of sequence and collaboration diagrams.

# Interaction diagram

It is the combination of sequence and collaboration diagram. It is used to depict the flow of events in the system over a timeline. The interaction diagram is a dynamic model which shows how the system behaves during dynamic execution.

# State chart diagram:

It consists of state, events and activities. State diagrams are a familiar technique to describe the behavior of a system. They describe all of the

possible states that a particular object can get into and how the object's state changes as a result of events that reach the object. In most OO techniques, state diagrams are drawn for a single class to show the lifetime behavior of a single object.

# Activity diagram:

It shows organization and their dependence among the set of components. These diagrams are particularly useful in connection with workflow and in describing behavior that has a lot of parallel processing. An activity is a state of doing something: either a real-world process, or the execution of a software routine.

# Implementation diagram:

It shows the implementation phase of the systems development, such as the source code structure and the run-time implementation structure. These are relatively simple high level diagrams compared to the others seen so far. They are of two sub-diagrams, the component diagram and the deployment diagram.

# Component diagram:

These are organizational parts of a UML model. These are boxes to which a model can be decomposed. They show the structure of the code itself. They model the physical components such as source code, user interface in a design. It is similar to the concept of packages.

# Deployment diagram:

The deployment diagram shows the structure of the runtime system. It shows the configuration of runtime processing elements and the software components that live in them. They are usually used in conjunction with deployment diagrams to show how physical modules of code are distributed on the system.

# NOTATION ELEMENTS:

These are explanatory parts of UML model. They are boxes which may apply to describe and remark about any element in the model. They

provide the information for understanding the necessary details of the diagrams.

# Relations in the UML:

are:

These are four kinds of relationships used in an UML diagram, they

* Dependency
* Association
* Generalization
* Realization

# Dependency:

It is a semantic relationship between two things in which a change one thing affects the semantics of other things. Graphically a dependency is represented by a non-continuous line.

# Association:

It is a structural relationship that describes asset of links

. A link is being connected among objects. Graphically association is represented as a solid line possibly including label.

# Generalization:

It is a specialized relationship in which the specialized elements are substitutable for object of the generalized element. Graphically it is a solid line with hollow arrow head parent.

# Realization:

It is a semantic relation between classifiers. Graphically it is represented as a cross between generalization and dependency relationship.

# Where UML can be used:

UML is not limited to modeling software. In fact it is expressive to model non-software such as to show in structure and behavior of health case system and to design the hardware of the system.

# Conceptual model be UML:

UML you need to form the conceptual model of UML. This requires three major elements:

* + UML basic building blocks.
  + Rules that dictate how this building blocks are put together.
  + Some common mechanism that apply throughout the language.

Once you have grasped these ideas, you may be able to read. UML create some basic ones. As you gain more experience in applying conceptual model using more advanced features of this language.

# Building blocks of the UML:

The vocabulary of UML encompasses these kinds of building blocks.

# Use CASE definition:

**Description:**

A use case is a set of scenarios tied together by a common user goal. A use case is a behavioral diagram that shows a set of use case actions and their relationships.

# Purpose:

The purpose of use case is login and exchange messages between sender and receiver (Email client).

# Main flow:

First, the sender gives his id and enters his login. Now, he enters the message to the receiver id.

# Alternate flow:

If the username and id by the sender or receiver is not valid, the administrator will not allow entering and “Invalid password” message is displayed.

# Pre-condition:

A person has to register himself to obtain a login ID.

# Post-condition:

The user is not allowed to enter if the password or user name is not

valid.

# Class diagram:

**Description:**

* A class diagram describes the type of objects in system and various kinds of relationships that exists among them.
* Class diagrams and collaboration diagrams are alternate representations of object models.

During analysis, we use class diagram to show roles and responsibilities of entities that provide email client system behaviors design. We use to capture the structure of classes that form the email client system architecture.

# A class diagram is represented as:

<<Class name>>

<<Attribute 1>>

<<Attribute n>>

<<Operation ()>>

# Relationship used:

A change in one element affects the other

# Generalization:

It is a kind of relationship

# State chart:

**Description:**

* The state chart diagram made the dynamic behavior of individual classes.
* State chart shows the sequences of states that an object goes through events and state transitions.
* A state chart contains one state ‘start’ and multiple ‘end’ states.

The important objectives are:

# Decision:

It represents a specific location state chart diagram where the work flow may branch based upon guard conditions.

# Synchronization:

It gives a simultaneous workflow in a state chart diagram. They visually define forks and joints representing parallel workflow.

# Forks and joins:

* + A fork construct is used to model a single flow of control.
  + Every work must be followed by a corresponding join.
  + Joints have two or more flow that unit into a single flow.

# State:

A state is a condition or situation during a life of an object in which it satisfies condition or waits for some events.

Transition:

It is a relationship between two activities and between states and activities.

# Start state:

A start state shows the beginning of a workflow or beginning of a state machine on a state chart diagram.

# End state:

It is a final or terminal state.

# Activity diagram Description:

Activity diagram provides a way to model the workflow of a development process. We can also model this code specific information such as class operation using activity diagram. Activity diagrams can model different types of diagrams. There are various tools involved in the activity diagram.

# Activity:

An activity represents the performance of a task on duty. It may also represent the execution of a statement in a procedure.

# Decision:

A decision represents a condition on situation during the life of an object, which it satisfies some condition or waits for an event.

# Start state:

It represents the condition explicitly the beginning of a workflow on an activity.

# Object flow:

An object on an activity diagram represents the relationship between activity and object that creates or uses it.

# Synchronization:

It enables us to see a simultaneous workflow in an activity.

# End state:

An end state represents a final or terminal state on an activity diagram or state chart diagram.

# Sequence diagram:

**Description:**

A sequence diagram is a graphical view of scenario that shows object interaction in a time based sequence what happens first what happens next. Sequence diagrams are closely related to collaboration diagram.

The main difference between sequence and collaboration diagram is that sequence diagram show time based interaction while collaboration diagram shows objects associated with each other.

The sequence diagram for the e-mail client system consists of the following objectives:

# Object:

An object has state, behavior and identity. An object is not based is referred to as an instance.

The various objects in e-mail client system are:

* + - User
    - Website
    - Login
    - Groups

# Message icon:

A message icon represents the communication between objects indicating that an action will follow. The message icon is the horizontal solid arrow connecting lifelines together.

# Collaboration diagram:

**Description:**

Collaboration diagram and sequence diagrams are alternate representations of an interaction. A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction.

Collaboration diagram shows object s, their links and their messages. They can also contain simple class instances and class utility instances.

During, analysis indicates the semantics of the primary and secondary interactions. Design, shows the semantics of mechanisms in the logical design of system.

Toggling between the sequence and collaboration diagrams

When we work in either a sequence or collaboration diagram, it is possible to view the corresponding diagram by pressing F5 key.

# CONCLUSION:

Thus the study for case tools was done.

USECASE DIAGRAM:

AIM: To identify actors associated with blood bank management and to generate a use case diagram affiliated with the blood bank management.

PURPOSE:

Actors:

1.Donor

2.Patient

3.Management

Donor: Donor is the one who donates the blood. Donor is the root node for the entire blood bank management tree.

Use cases=3

1. sign up at website.
2. Get appointment at blood bank
3. Donates blood.

Donor is directly related with management.

Type of relation: Associative

Patient: Patient is the one who needs blood from blood bank.

Use cases=1

1. Make Payment and Receive specimen.

Patient is directly related with management.

Type of relation: Associative

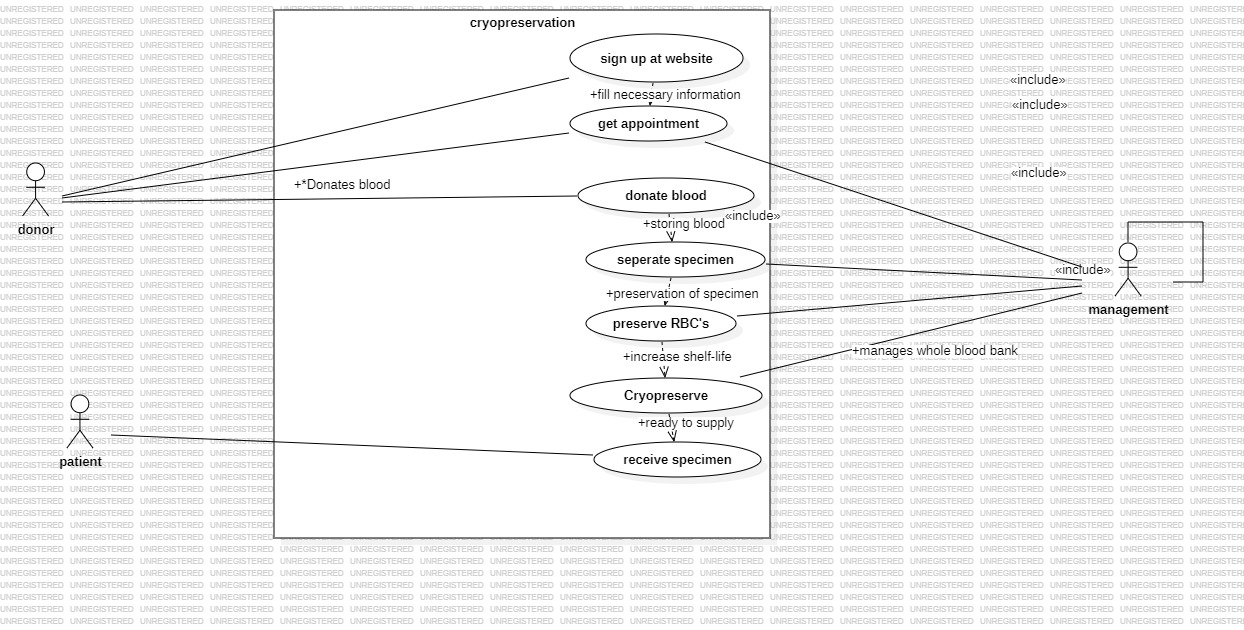
Management: Management is the organization link the patient and donor, it stores the received blood from donor and arranges it to patient when needed.

Use cases=4

1. Give Appointment.
2. Separate specimen.
3. Preserves RBC.
4. Cryopreserve.

Management is directly related with both patient and donor.

Type of relation: Associative



CLASS DIAGRAM:

AIM: To identify the required procedures in blood bank management.

Purpose:

Classes:

1.Blood management

2.Donor

3.Patient

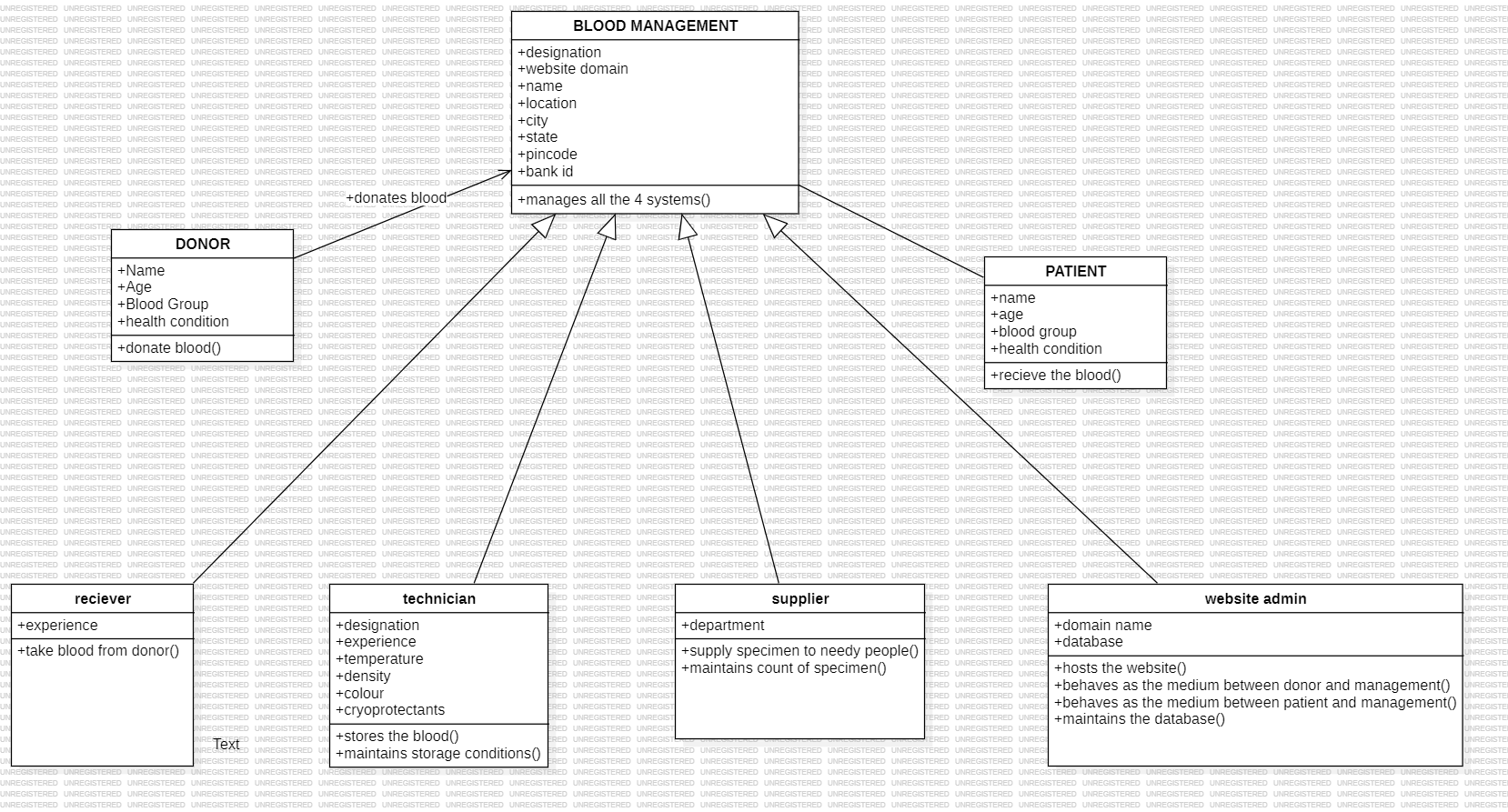
4.Receiver

5.technician

6.supplier

7.web admin

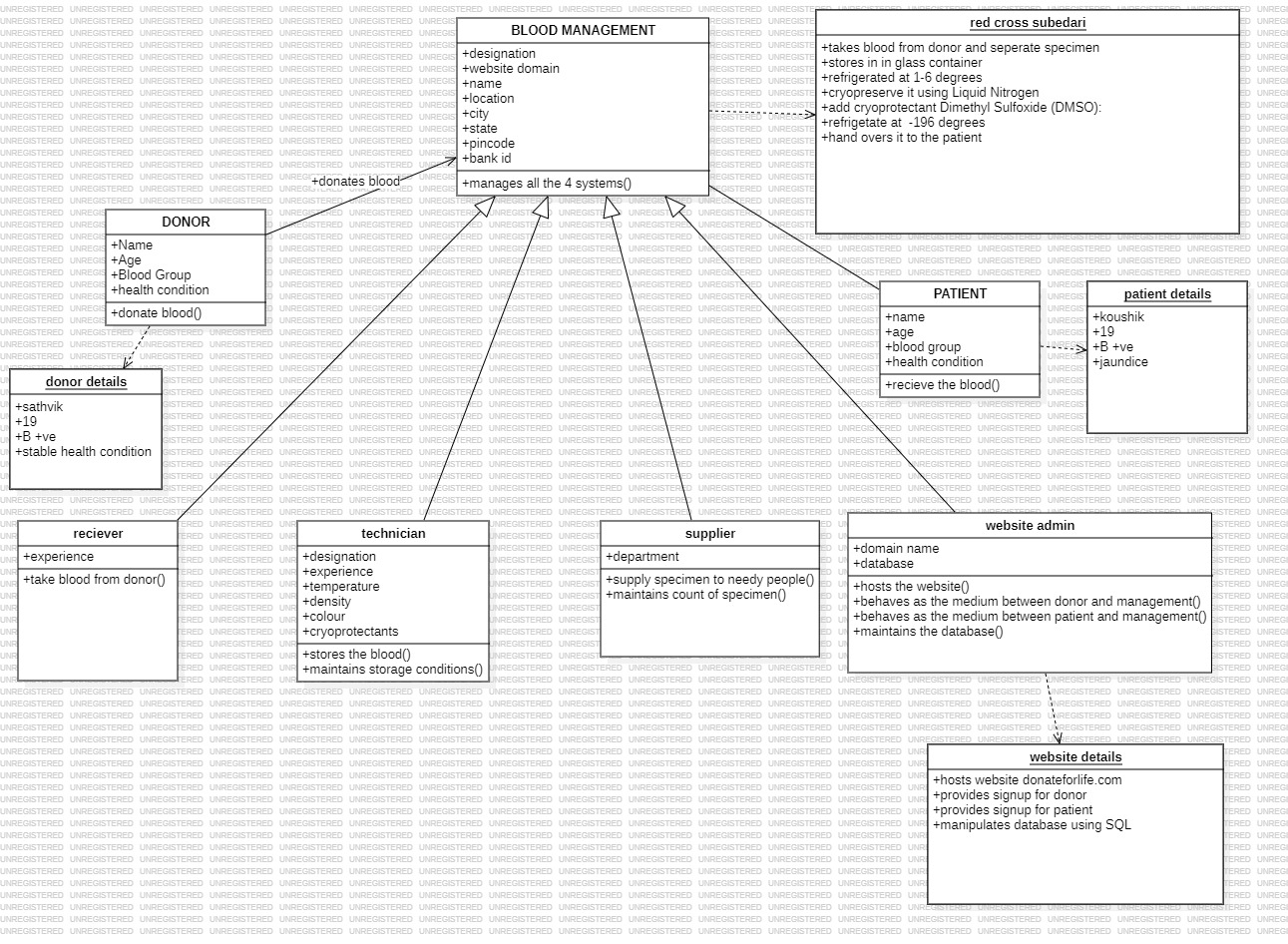
All the classes are associated with Main class Blood Management



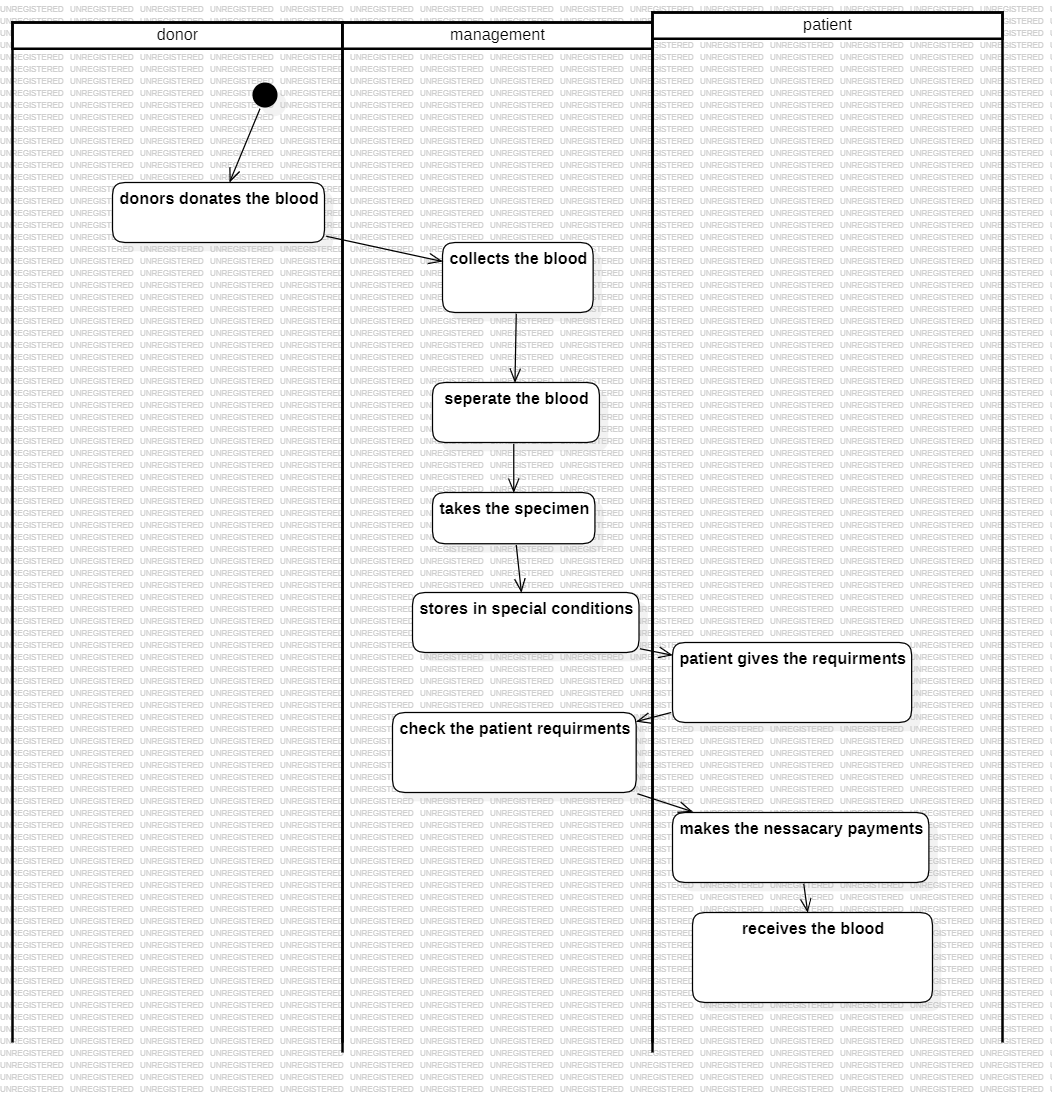
OBJECT DIAGRAM:

AIM: To define the object terminology to class.

Purpose:



1. **ACTIVITY DIAGRAM:**



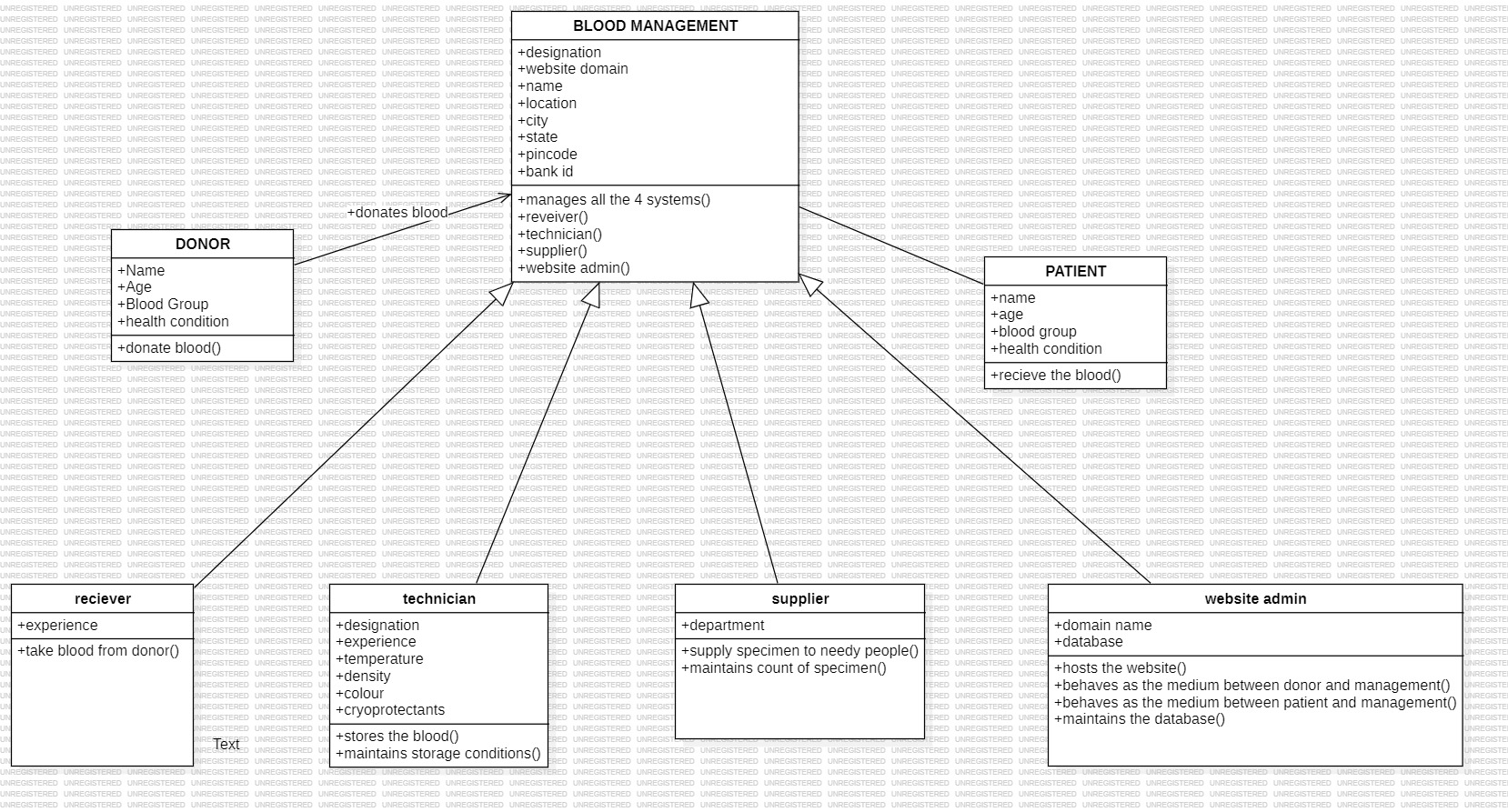
# Fig.4.1. ACTIVITY DIAGRAM FOR REGISTER

1. **CLASS DIAGRAM:**

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

The Passport Automation system class diagram consists of four classes Passport Automation System

* 1. New registration
  2. Gender
  3. Application Status
  4. Admin authentication
  5. Admin Panel



# Fig.5. CLASS DIAGRAM FOR PASSPORT AUTOMATION SYSTEM

1. **INTERACTION DIAGRAM:**

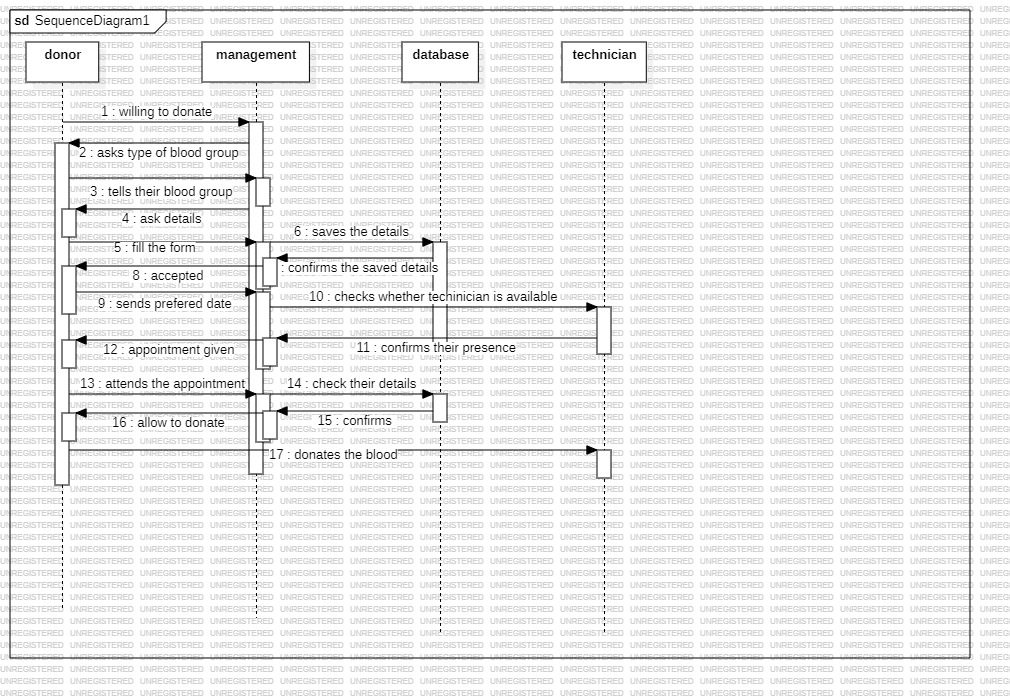
A sequence diagram represents the sequence and interactions of a given USE-CASE or scenario. Sequence diagrams can capture most of the information about the system. Most object to object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the “form” object is requesting an operation be performed by the “to” object. The “to” object performs the operation using a method that the class contains.

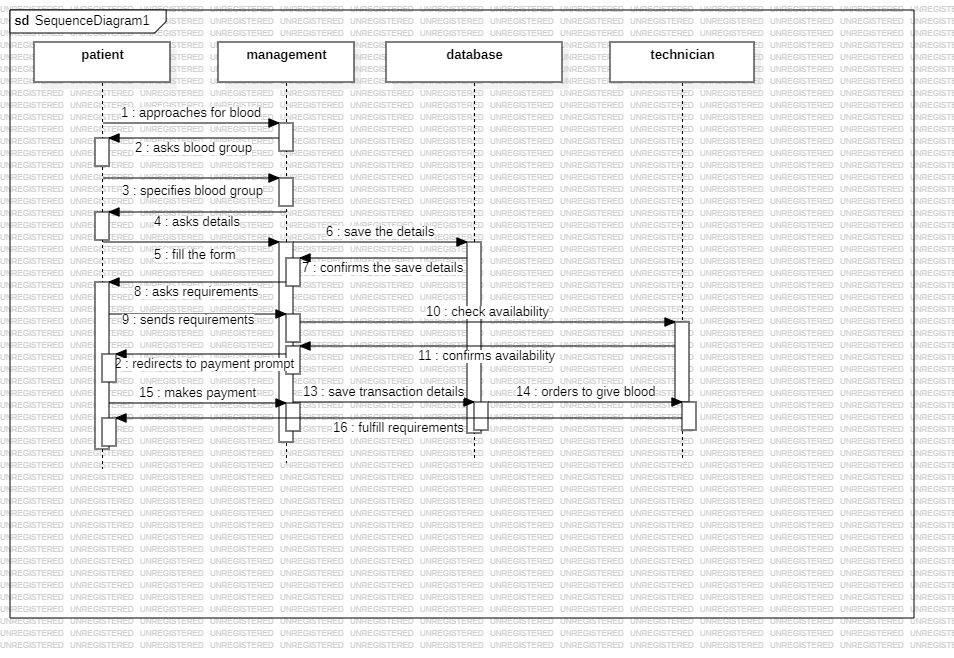
It is also represented by the order in which things occur and how the objects in the system send message to one another.

The sequence diagram for each USE-CASE that exists when a user administrator, check status and new registration about passport automation system are given.

**Fig.6.1.SEQUENCE DIAGRAM FOR DONOR:**



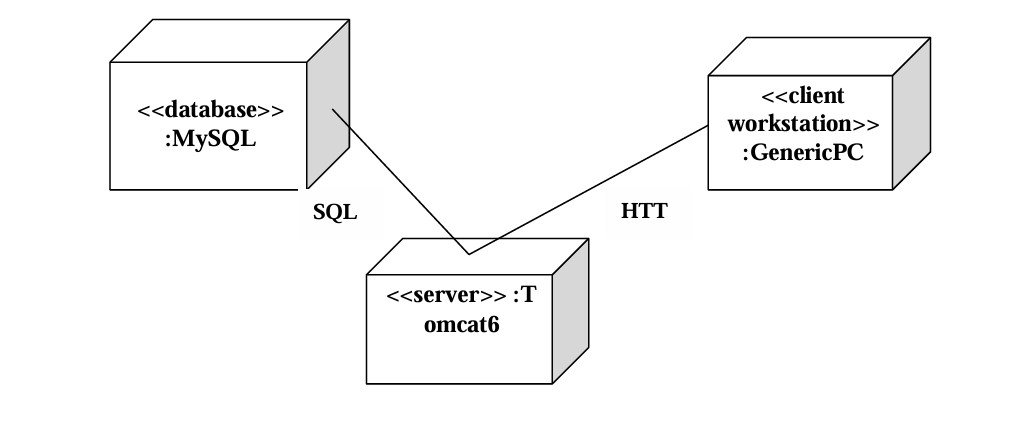
**Fig.6.2. COLLABORATION DIAGRAM FOR PATIENT:**



**(VII) PARTIAL LAYERD LOGICAL ARCHITECTURE DIAGRAM**

**(VIII) DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM**

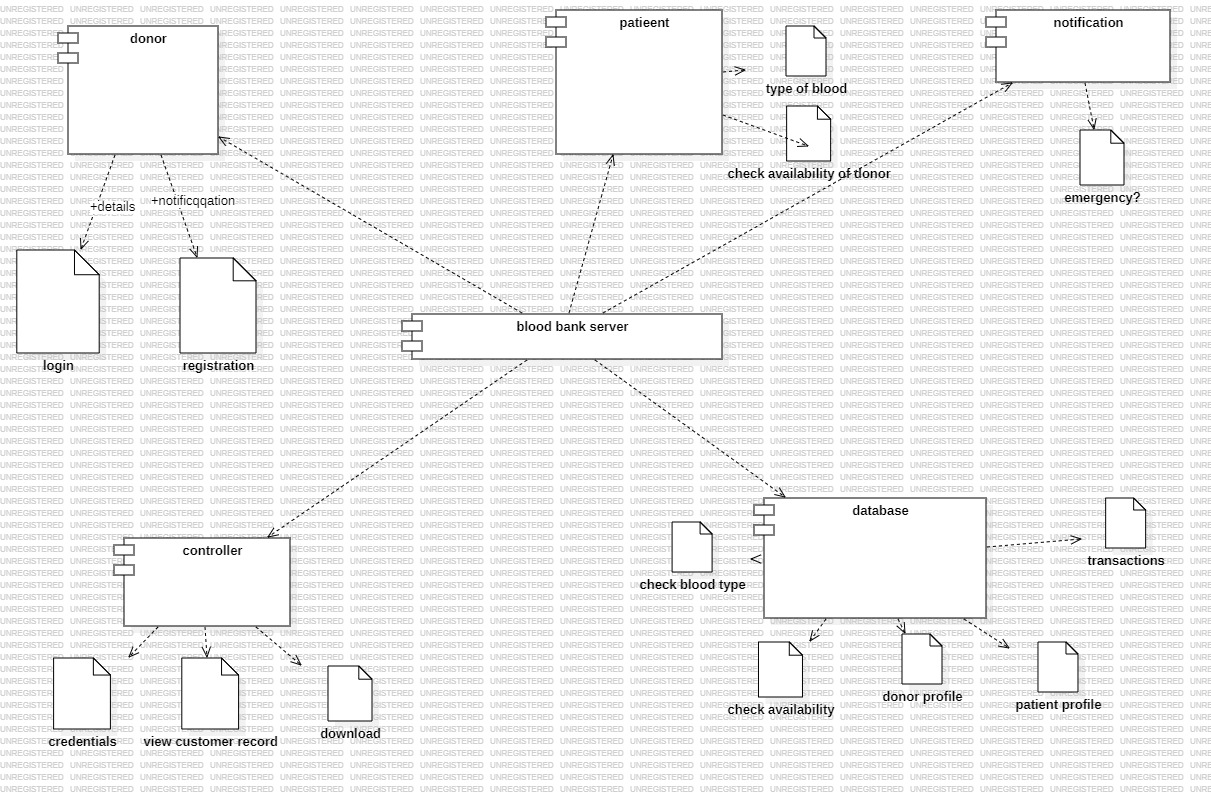
Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.



DEPLOYMENT DIAGRAM

COMPONENT DIAGRAM :

Component diagrams are used to visualize the organization and relationship among components in system.



## 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.>

## Overview

<Provide overall general information regarding the product in terms of marketing and advertisement which includes its utility in current prospect and its future use and state.>

# Overall Description

## 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.>

## Product Functions

<Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Details will be provided in Section 3, so only a high-level summary 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 a class diagram, is often effective.>

## User 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 favored user classes from those who are less important to satisfy.>

## General 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).>

## 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).>

# Specific Requirements

## External Interface Requirements

### 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.>

### 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.>

### 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.>

### Communication 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.>

## Functional Requirements

<In this section, the functional capabilities of the system are describe. In this organization, the functional capabilities for all the modes of operation of the software are given. For each functional requirement, the required inputs, desired outputs, and processing requirements will have to be specified. For the inputs, the source of the inputs, the units of measure, valid ranges, accuracies, etc. have to be specified. For specifying the processing, all operations that need to be performed on the input data and any intermediate data produced should be specified. This includes validity checks on inputs, sequence of operations, responses to abnormal situations, and methods that must be used in processing to transform the inputs into corresponding outputs. Note that no algorithms are generally specified, only the relationship between the inputs and the outputs (which may be in the form of an equation or a formula) so that the designer can design an algorithm to produce the outputs from the inputs. For outputs, the destination of outputs, units of measure, range of valid outputs, error messages, etc. all have to be specified.>

### Mode 1

#### Functional Requirement 1.1

:

#### Functional Requirement 1.*n*

:

### Mode *m*

#### Functional Requirement *m*.1

:

#### Functional Requirement *m*.*n*

## Performance Requirements

<The performance section should specify both static and dynamic performance requirements. All factors that constrain the system design are described in the performance constraints section. The attributes section specifies some of the overall attributes that the system should have. Any requirement not covered under these is listed under other requirements. Design constraints specify all the constraints imposed on design (e.g. security, fault tolerance, and standards compliance. 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.>

## Design constraints

<Design constraints mention the issue related to organization specific.>

## 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.>

## 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.>