ABSTRACT

Abstract:

The main objective of this system is to provide a mediation interface among the factories, suppliers and the mediators who will carry the stock and store in their own GODOWNs and forward them to the suppliers when required.

Existing System:

Profile:

Some of the factories will maintain their product in their own GODOWNs until those will be delivered to the suppliers. Whenever the supplier needs the stock then the factories will forward the requested stock to the suppliers.

Existing System:

In existing system the factories has to maintain lot of GODOWNs for storing the stock. For that they need to invest a lot on that. These are unnecessary investments and they should do more work for that. It will require mediation between the factory to the supplier for saving time and space.

Proposed System:

In proposed system there is a mediatory GODOWNs management companies. They will carry the stock from the factories and forward the stock to the suppliers. They can check the availability of the required space in online and send the stock. It's an online application which automates all the operations and maintains a centralized database.

1 SYSYTEM ANALYSIS

1.1 STUDY OF THE SYSTEM

To provide flexibility to the users, the interfaces have been developed that are accessible through a browser. The GUI'S at the top level have been categorized as

1. Administrative user interface
2. The operational or generic user interface

The 'administrative user interface' concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. These interfaces help the administrators with all the transactional states like Data insertion, Data deletion and Date updation along with the extensive data search capabilities.The 'operational or generic user interface' helps the end users of the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information in a customized manner as per the included flexibilities

1.2 INPUT & OUTPOUT REPRESENTETION

Input design is a part of overall system design. The main objective during the input design is as given below:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable and understood by the user.

INPUT STAGES:

The main input stages can be listed as below:

- Data recording
- Data transcription
- Data conversion

- Data verification
- Data control
- Data transmission
- Data validation
- Data correction

INPUT TYPES:

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

- External inputs, which are prime inputs for the system.
- Internal inputs, which are user communications with the system.
- Operational, which are computer department's communications to the system?
- Interactive, which are inputs entered during a dialogue.

INPUT MEDIA:

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

- Type of input
- Flexibility of format
- Speed
- Accuracy
- Verification methods
- Rejection rates
- Ease of correction
- Storage and handling requirements
- Security
- Easy to use
- Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

OUTPUT DESIGN:

In general are:

- External Outputs whose destination is outside the organization.
- Internal Outputs whose destination is with in organization and they are the User's main interface with the computer. Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provide a permanent copy of the results for later consultation. The various types of outputs
- Operational outputs whose use is purely with in the computer department.
- Interface outputs, which involve the user in communicating directly with the system.

OUTPUT DEFINITION

The outputs should be defined in terms of the following points:

- Type of the output
- Content of the output
- Format of the output
- Location of the output
- Frequency of the output
- Volume of the output
- Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

For Example

- Will decimal points need to be inserted
- Should leading zeros be suppressed.

OUTPUT MEDIA:

In the next stage it is to be decided that which medium is the most appropriate for the output. The main considerations when deciding about the output media are:

- The suitability for the device to the particular application.
- The need for a hard copy.
- The response time required.
- The location of the users
- The software and hardware available.

Keeping in view the above description the project is to have outputs mainly coming under the category of internal outputs. The main outputs desired according to the requirement specification are:

The outputs were needed to be generated as a hard copy and as well as queries to be viewed on the screen. Keeping in view these outputs, the format for the output is taken from the outputs, which are currently being obtained after manual processing. The standard printer is to be used as output media for hard copies.

1.3 PROCESS MODEL USED WITH JUSTIFICATION

SDLC (Spiral Model):

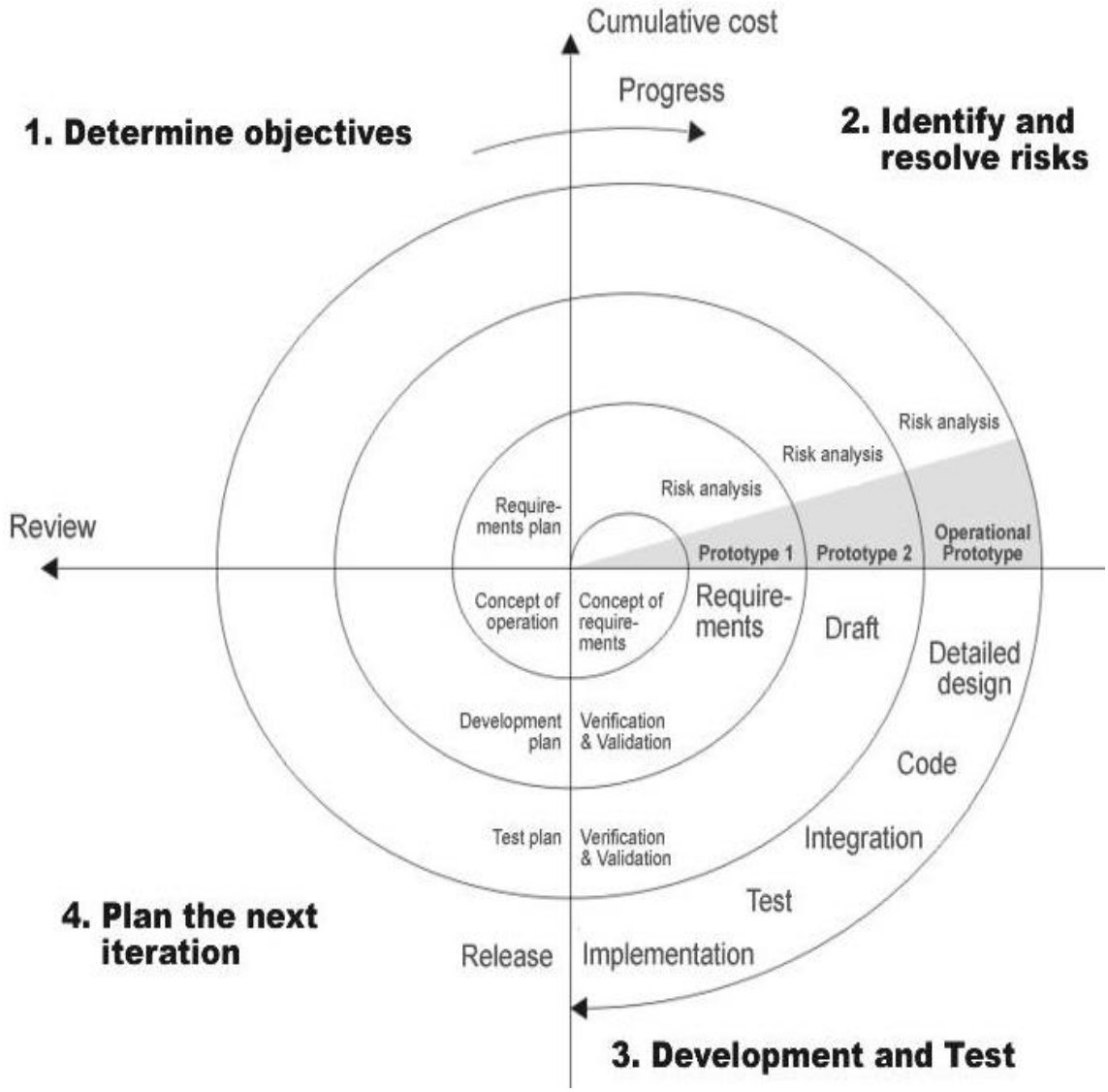


Fig 1.3 SDLC

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

1.4 Stages in SDLC:

- Requirement Gathering
- Analysis
- Designing
- Coding
- Testing
- Maintenance

1.4.1Requirements Gathering stage:

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.

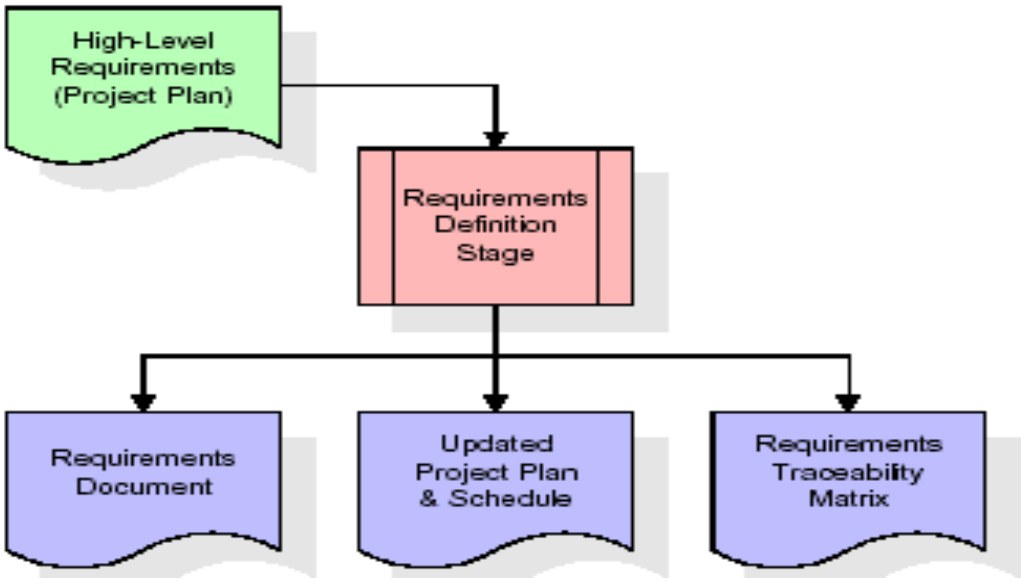


Fig 1.4 Requirements gathering stage

These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term *requirements traceability*.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

- Feasibility study is all about identification of problems in a project.
- No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
- Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator

1.4.2Analysis Stage:

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.

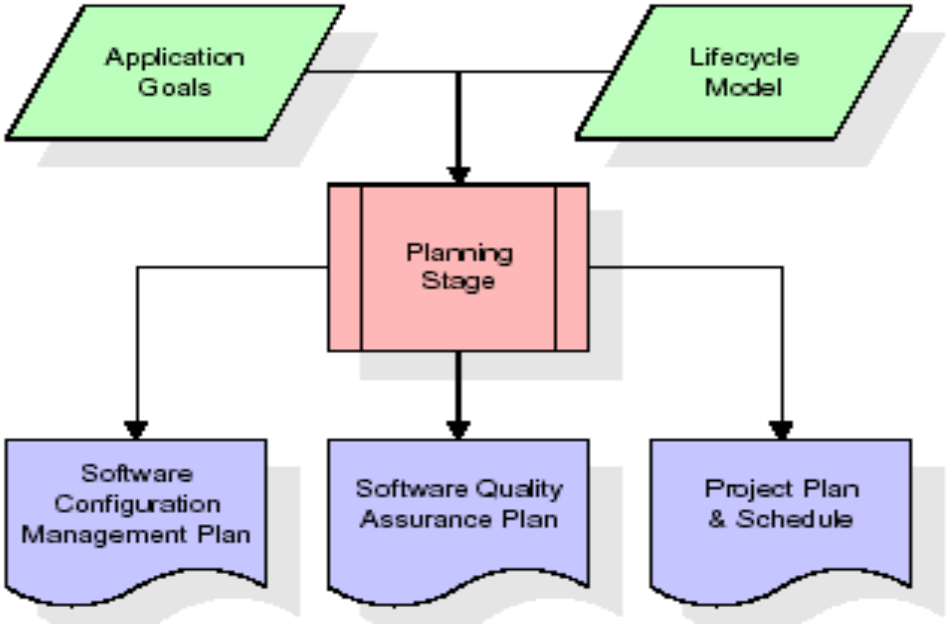


Fig 1.4.2 Analysis Stage

The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information

for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

1.4.3Designing Stage:

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

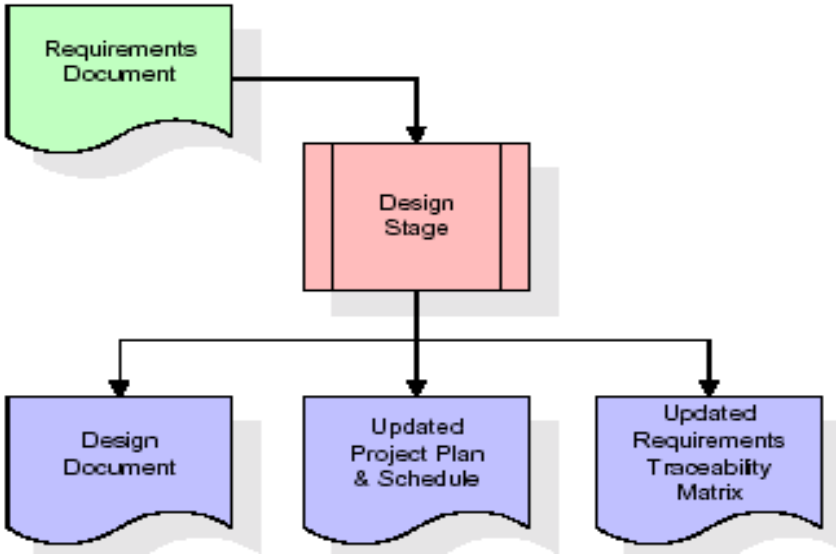


Fig 1.4.3 Dominating stage

When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

1.4.4 Development (Coding) Stage:

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.

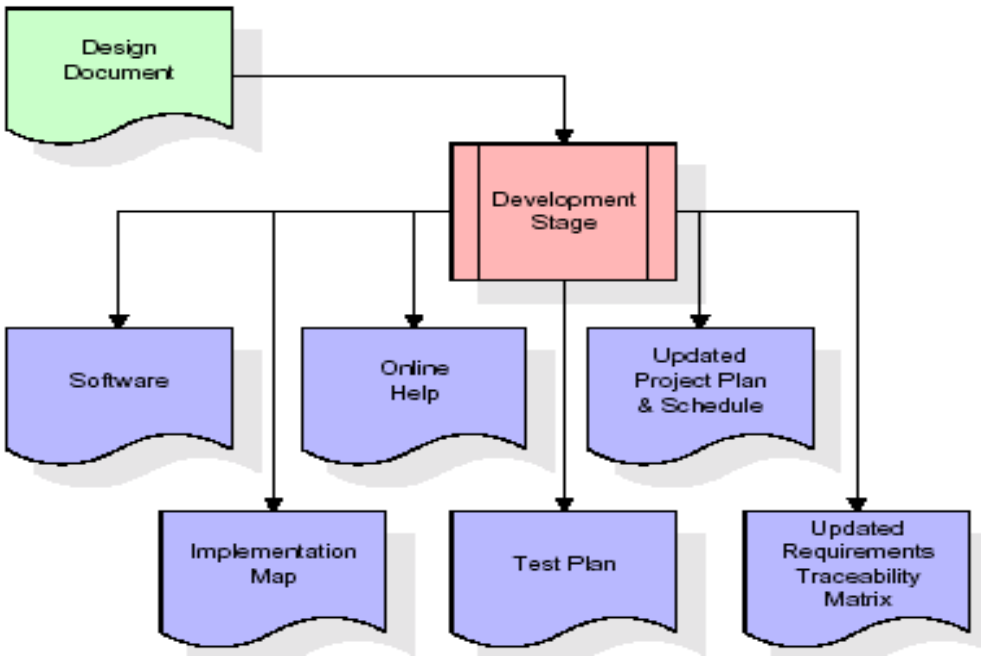


Fig 1.4.4 Development stage

The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

1.4.5 Integration & Test Stage:

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.

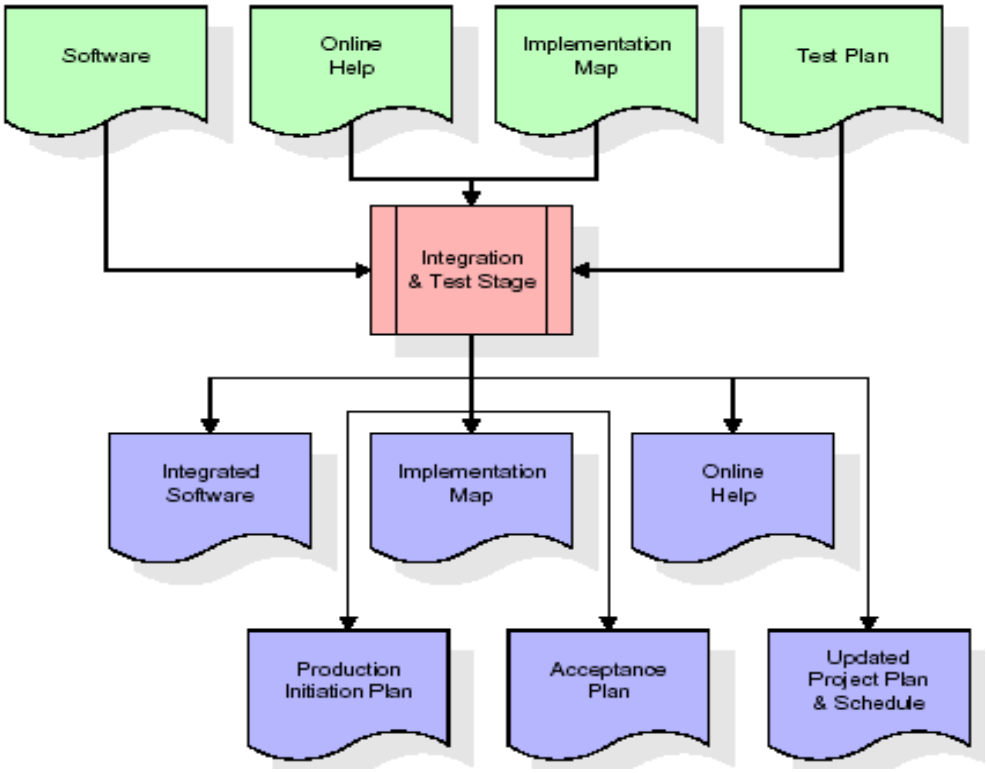


Fig 1.4.5 Integration stage

The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

- **Installation & Acceptance Test:**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

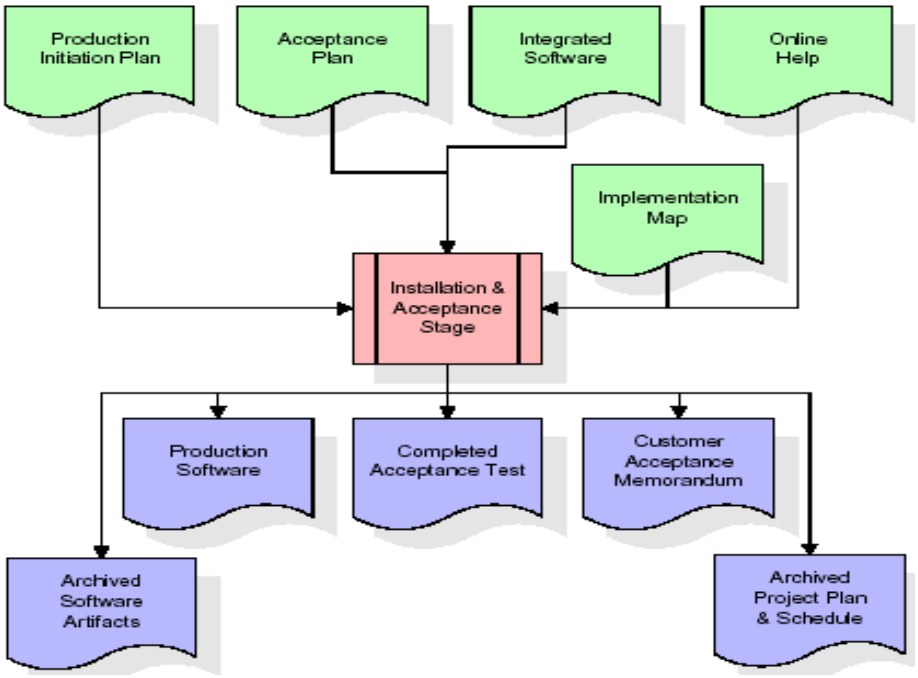


Fig 1.4.5 Installation and acceptance test

The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.1.

1.4.6 Maintenance

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will under go training on that particular assigned category.

For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

1.5 SYSTEM ARCHITECTURE

Architecture flow:

Below architecture diagram represents mainly flow of requests from users to database through servers. In this scenario overall system is designed in three tires separately using three layers called presentation layer, business logic layer and data link layer. This project was developed using 3-tire architecture.

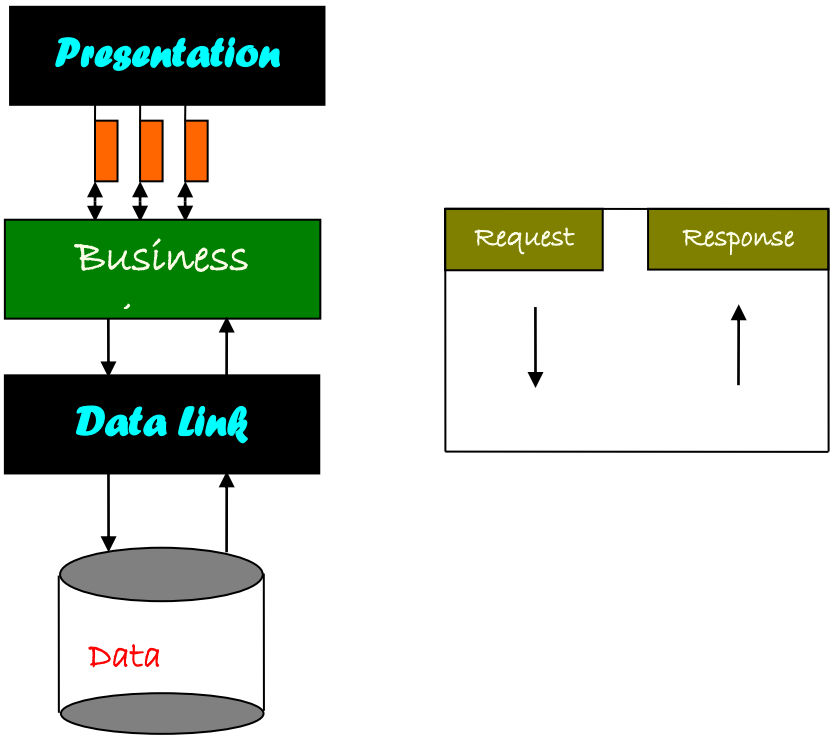


Fig 1.5 system architecture

2.FEASIBILITY STUDY

Feasibility Study:

Preliminary investigation examines project feasibility; the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All systems are feasible if they are given unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operation Feasibility

Economical Feasibility

2.1 TECHNICAL FEASIBILITY

The technical issue usually raised during the feasibility stage of the investigation includes the following:

- Does the necessary technology exist to do what is suggested?
- Do the proposed equipments have the technical capacity to hold the data required to use the new system?
- Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
- Can the system be upgraded if developed?

Are there technical guarantees of accuracy, reliability, ease of access and data security?

2.2 OPERATIONAL FEASIBILITY

User-friendly

Customer will use the forms for their various transactions i.e. for adding new routes, viewing the routes details. Also the Customer wants the reports to view the various transactions based on the constraints. Theses forms and reports are generated as user-friendly to the Client.

Reliability

The package wills pick-up current transactions on line. Regarding the old transactions, User will enter them in to the system.

Security

The web server and database server should be protected from hacking, virus etc

Portability

The application will be developed using standard open source software (Except Oracle) like Java, tomcat web server, Internet Explorer Browser etc these software will work both on Windows and Linux o/s. Hence portability problems will not arise.

Availability

This software will be available always.

Maintainability

The system called the ewheelz uses the 2-tier architecture. The 1st tier is the GUI, which is said to be front-end and the 2nd tier is the database, which uses My-Sql, which is the back-end.

The front-end can be run on different systems (clients). The database will be running at the server. Users access these forms by using the user-ids and the passwords.

2.3 ECONOMIC FEASILITY

The computerized system takes care of the present existing system's data flow and procedures completely and should generate all the reports of the manual system besides a host of other management reports.

It should be built as a web based application with separate web server and database server. This is required as the activities are spread through out the organization customer wants a centralized database. Further some of the linked transactions take place in different locations.

Open source software like TOMCAT, JAVA, Mysql and Linux is used to minimize the cost for the Customer.

3.REQUIREMENTS SPECIFICATIONS:

3.1 FUNCTIONAL REQUIREMENTS SPECIFICATION

This application consists following modules.

Modules:

1. Administrator Module.
2. GODOWN Manager Module.
3. Vendor/Supplier Module.
4. Accountant Module.
5. Inventory Manager Module.

Administrator Module:

- He is the owner of the GODOWNs maintaining organization.
- He able to create, update and delete the GODOWNs.

- He able to create, update and delete the GODOWN managers, inventory managers and accountants.
- He able to assign the managers to particular GODOWNs.
- He able to generate the reports based on different criteria and export to XL sheet.

GODOWN Manager Module:

- He will give the access to the vendors as well as suppliers.
- Get the stock/goods from the vendors.
- Maintain the inwards information.
- Get the request from the supplier and respond to that request.
- Generate the reports based on inwards information and export to XL sheet.

Inventory Manager Module:

- Able to access the GODOWN details like space availability.
- Get the request from the vendor and respond to that request.
- Maintain the GODOWN information like space, stock, etc…

Accountant Module:

- Will access the inwards as well as outwards information.
- Generate the bills for the inwards stock and outwards stock.
- Maintain the payments information.

Vendor/Supplier Module:

- Both Vendors as well as Suppliers are able to register in this application.
- Vendor can check the space availability.
- Vendor can give the request for available space.
- Supplier can check whether the stock which he ordered is available or not

- Supplier can give request for the required stock/goods.
- Vendor as well as Supplier can make payments for the usage of the particular GODOWNs.

Both can generate reports like the manufactured goods, delivered stock and export to XL sheet.

3.2 SOFTWARE REQUIREMENTS:

- Technology : PYTHON
- Framework : Django
- Web Technologies : Html, JavaScript, CSS
- IDE : Pycharm
- Web Server : Django Server
- Database : SQLite3

3.3 HARDWARE REQUIREMENTS:

- Hardware : Pentium
- Speed : 1.1 GHz
- RAM : 1GB
- Hard Disk : 20 GB

3.4.1. INTRODUCTION TO PYTHON

Python is a general purpose programming language. Hence, you can **use** the programming language for developing both desktop and web applications. Also, you can **use Python** for developing complex scientific and numeric applications. **Python** is designed with features to facilitate data analysis and visualization

3.4.2 DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source

3.4.3 SQLITE

Dr.E.F.Codd's Rules

These rules are used for valuating a product to be called as relational database management systems. Out of 12 rules, a RDBMS product should satisfy at least 8 rules + rule called rule 0 that must be satisfied.

RULE 0: Foundation Rule

For any system to be advertised as, or claimed to be relational DBMS should manage database with in it self, with out using an external language.

RULE 1: Information Rule

All information in relational database is represented at logical level in only one way as values in tables.

RULE 2: Guaranteed Access

Each and every data in a relational database is guaranteed to be logically accessibility by using to a combination of table name, primary key value and column name.

RULE 3: Systematic Treatment of Null Values

Null values are supported for representing missing information and inapplicable information. They must be handled in systematic way, independent of data types.

RULE 4: Dynamic Online Catalog based Relation Model

The database description is represented at the logical level in the same way as ordinary data so that authorized users can apply the same relational language to its interrogation as they do to the regular data.

RULE 5: Comprehensive Data Sub Language

A relational system may support several languages and various models of terminal use. However there must be one language whose statement can express all of the following:

Data Definitions, View Definitions, Data Manipulations, Integrity, Constraints, Authorization and transaction boundaries.

RULE 6: View Updating

Any view that is theoretical can be updatable if changes can be made to the tables that effect the desired changes in the view.

3.4.4 JAVA SCRIPT

RULE 7: High level Update, Insert and Deete

The capability of handling a base relational or derived relational as a single operand applies not only retrieval of data also to its insertion, updating, and deletion

RULE 8: Physical Data Independence

Application program and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access method.

RULE 9: Logical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

RULE 10: Integrity Independence

Integrity constraints specific to particular database must be definable in the relational data stored in the catalog, not in application program.

RULE 11: Distributed Independence

Whether or not a system supports database distribution, it must have a data sub-language that can support distributed databases without changing the application program.

RULE 12: Non Sub-Version

If a relational system has low level language, that low language cannot use to subversion or by pass the integrity rules and constraints expressed in the higher level relational language.

3.4.5 HTML

Hypertext Markup Language(HTML), the languages of the world wide web(WWW), allows users to produces web pages that included text, graphics and pointer to other web pages (Hyperlinks).

HTML is not a programming language but it is an application of ISO Standard 8879,SGML(Standard Generalized Markup Language),but

Specialized to hypertext and adapted to the Web. The idea behind Hypertext one point to another point. We can navigate through the information based on out interest and preference. A markup language is simply a series of items enclosed within the elements should be displayed.

Hyperlinks are underlined or emphasized works that load to other documents or some portions of the same document.

Html can be used to display any type of document on the host computer, which can be geographically at a different location. It is a versatile language and can be used on any platform or desktop.

HTML provides tags(special codes) to make the document look attractive.

HTML provides are not case-sensitive. Using graphics,fonts,different sizes, color, etc.. can enhance the presentation of the document. Anything

That is not a tag is part of the document it self.

Basic Html Tags:

<!-- -->	Specific Comments.
<A>……</A>	Creates Hypertext links.
<B>……</B>	Creates hypertext links.
<Big>……..</Big>	Formats text in large-font
<Body>…….</Body>	contains all tags and text in the Html-document
<Center>……</Center>	Creates Text
<DD>……..</DD>	Definition of a term.
<TABLE>……</TABLE>	creates table

<Td>……..</Td> indicates table data in a table.

<Tr>……..</Tr> designates a table row

<Th>…….</Th> creates a heading in a table.

A D V A N T A G E S:-

- A HTML document is small and hence easy to send over the net.It is small because it does not include formatted information.
- HTML is platform independent

HTML tags are not case-sensitive

The Java Script Language

JavaScript is a compact , object-based scripting language for developing client and server internet applications. Netscape Navigator 2.0 interprets JavaScript statements embedded directly in an HTML page. and Livewire enables you to create server-based applications similar to common gateway interface(cgi) programs.

In a client application for Navigator, JavaScript statements embedded in an HTML Page can recognize and respond to user events such as mouse clicks form

Input, and page navigation.

For example, you can write a JavaScript function to verify that users enter valid information into a form requesting a telephone number or zip code . Without any network transmission, an Html page with embedded Java Script can interpret the entered text and alert the user with a message dialog if the input is invalid or you can use JavaScript to perform an action (such as play aaudio file, execute an applet, or communicate with a plug-in) in response to the user opening or exiting a page.

4.SYSTEM DESIGN

4.1 INTRODUCTION

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap and synergy with the disciplines of systems analysis, systems architecture and systems engineering.

4.2 UML DIAGRAMS

Unified Modeling Language:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

- User Model View
    i. This view represents the system from the users perspective.
    ii. The analysis representation describes a usage scenario from the end-users perspective.

- Structural model view
    i. In this model the data and functionality are arrived from inside the system.
    ii. This model view models the static structures.

- Behavioral Model View

  It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

- Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

Environmental Model View

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

- ✓ UML Analysis modeling, this focuses on the user model and structural model views of the system.
- ✓ UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user's point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

4.3. NORMALIZATION

A Database is a collection of interrelated data stored with a minimum of redundancy to serve many applications. The database design is used to group data into a number of tables and minimizes the artificiality embedded in using separate files. The tables are organized to:

- Reduced duplication of data.
- Simplify functions like adding, deleting, modifying data etc..,
- Retrieving data

- Clarity and ease of use
- More information at low cost

Normalization

Normalization is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints on the kind of functional dependencies that could be associated with the relation. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database.

First Normal Form:

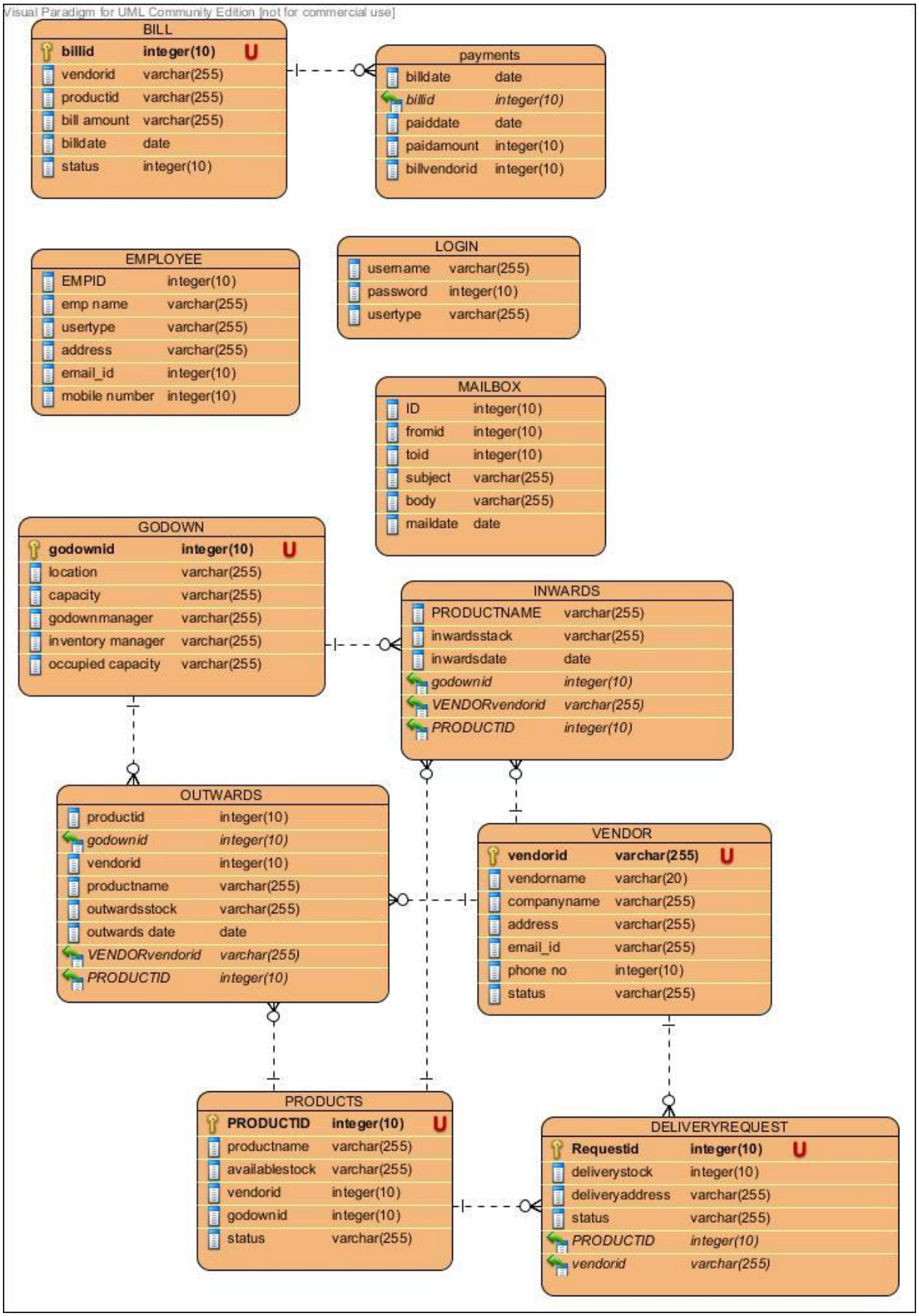
A relation R is in first normal form if and only if all underlying domains contained atomic values only.

Second Normal Form:

A relation R is said to be in second normal form if and only if it is first normal form and every non-key attribute is fully dependent on the primary key.

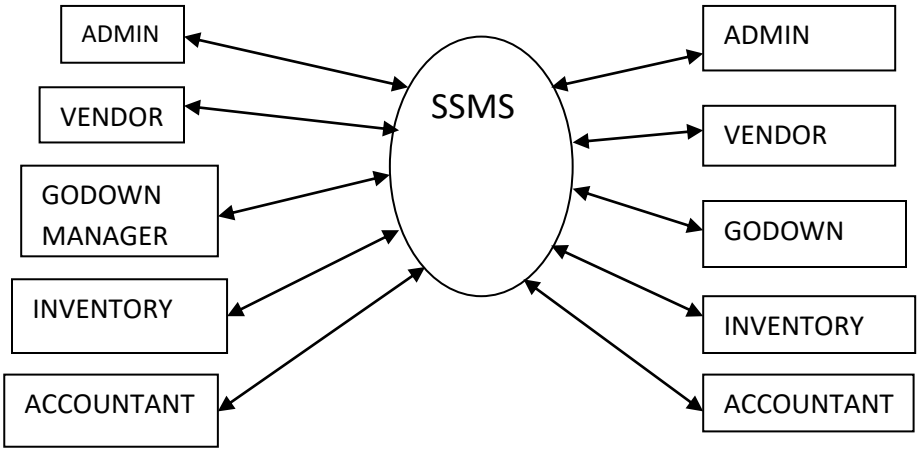
Third Normal Form :

A relation R is said to be in third normal form if and only if it is in second normal form and every non key attribute is non transitively depend on the primary key.

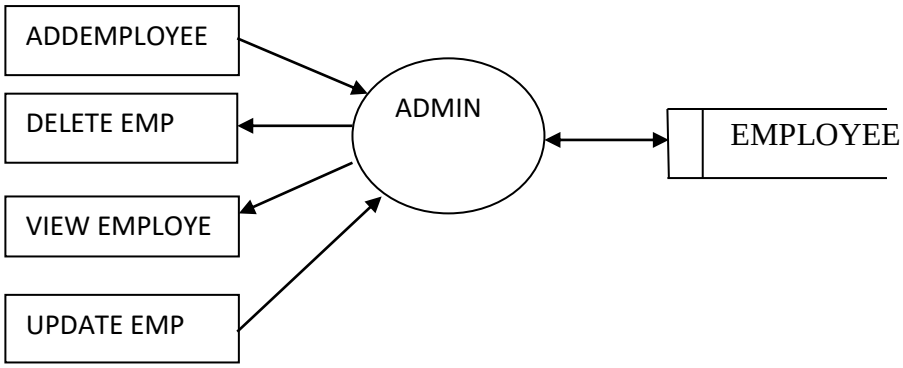

Visual Paradigm for UML Community Edition [not for commercial use]
BILL
billid integer(10) U
vendorid varchar(255)
productid varchar(255)
bill amount varchar(255)
billdate date
status integer(10)
payments
billdate date
billid integer(10)
paiddate date
paidamount integer(10)
billvendorid integer(10)
EMPLOYEE
EMPID integer(10)
emp name varchar(255)
usertype varchar(255)
address varchar(255)
email_id integer(10)
mobile number integer(10)
LOGIN
username varchar(255)
password integer(10)
usertype varchar(255)
MAILBOX
ID integer(10)
fromid integer(10)
toid integer(10)
subject varchar(255)
body varchar(255)
maildate date
GODOWN
godownid integer(10) U
location varchar(255)
capacity varchar(255)
godownmanager varchar(255)
inventory manager varchar(255)
occupied capacity varchar(255)
INWARDS
PRODUCTNAME varchar(255)
inwardsstack varchar(255)
inwardsdate date
godownid integer(10)
VENDORvendorid varchar(255)
PRODUCTID integer(10)
OUTWARDS
productid integer(10)
godownid integer(10)
vendorid integer(10)
productname varchar(255)
outwardsstock varchar(255)
outwards date date
VENDORvendorid varchar(255)
PRODUCTID integer(10)
VENDOR
vendorid varchar(255) U
vendorname varchar(20)
companyname varchar(255)
address varchar(255)
email_id varchar(255)
phone no integer(10)
status varchar(255)
PRODUCTS
PRODUCTID integer(10) U
productname varchar(255)
availablestock varchar(255)
vendorid integer(10)
godownid integer(10)
status varchar(255)
DELIVERYREQUEST
Requestid integer(10) U
deliverystock integer(10)
deliveryaddress varchar(255)
status varchar(255)
PRODUCTID integer(10)
vendorid varchar(255)

5. DFD:(DATA FLOW DIAGRAMS):

CONTEXT LEVEL-DFD:

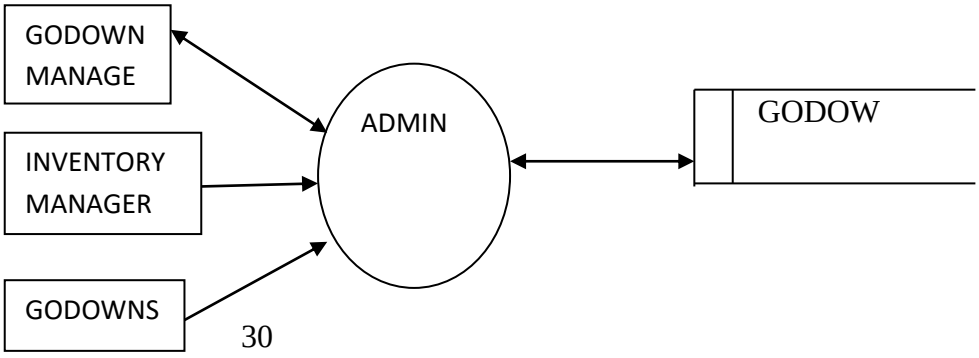


LEVEL-1:

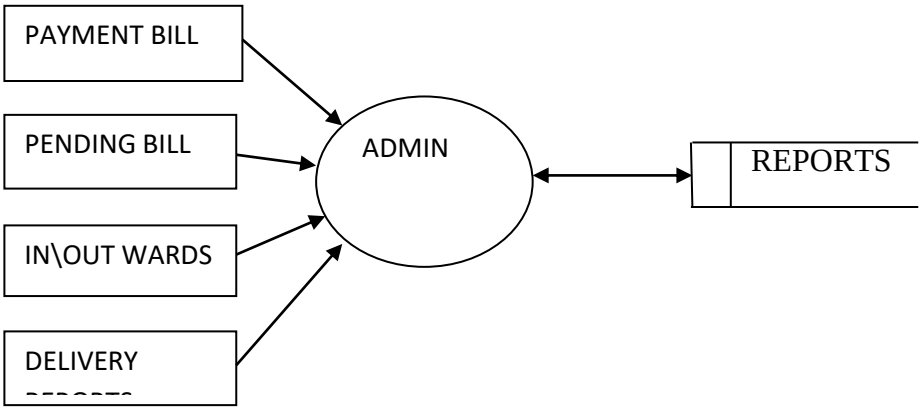


LEVEL-2

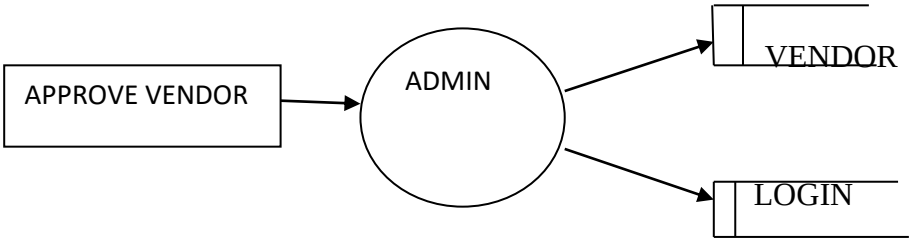
N



LEVEL-3:



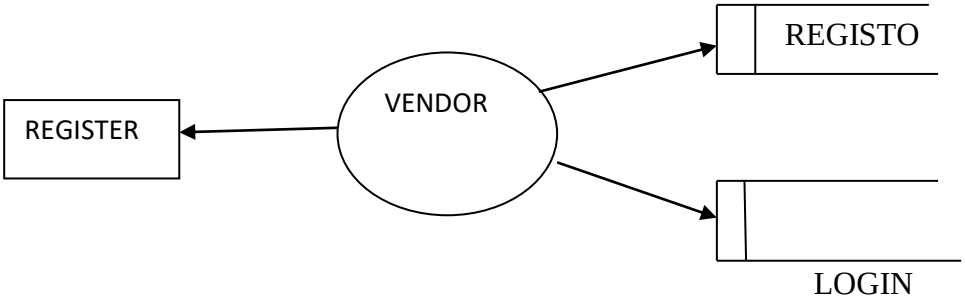
LEVEL-4:



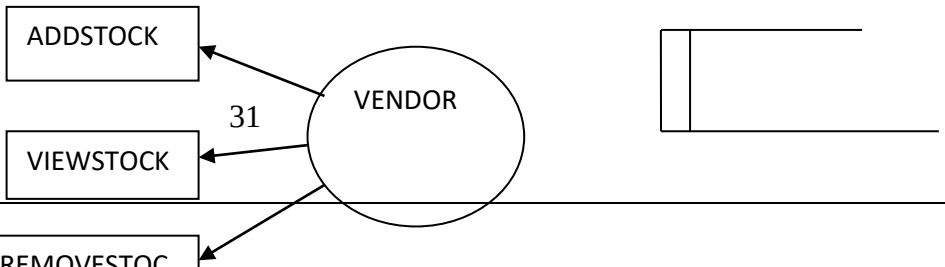
VENDOR-DFD:

LEVEL0:

R



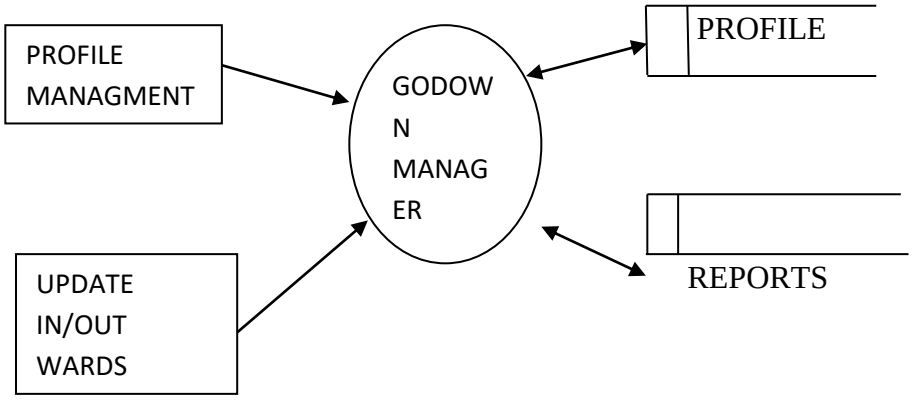
LEVEL1:



STOCK

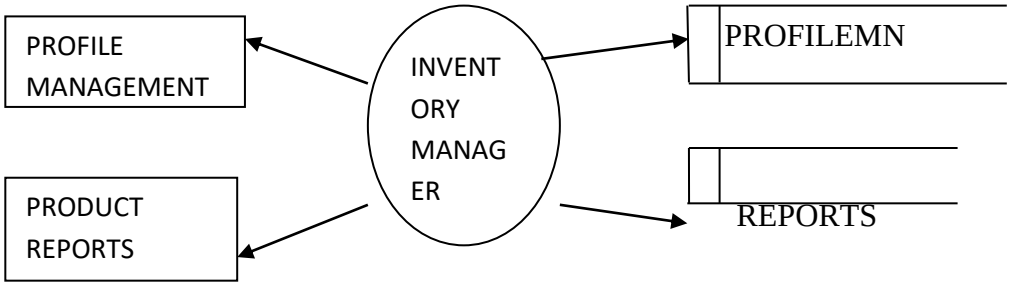
GODOWN MANAGER:

MANA



INVENTORY MANAGER-DED:

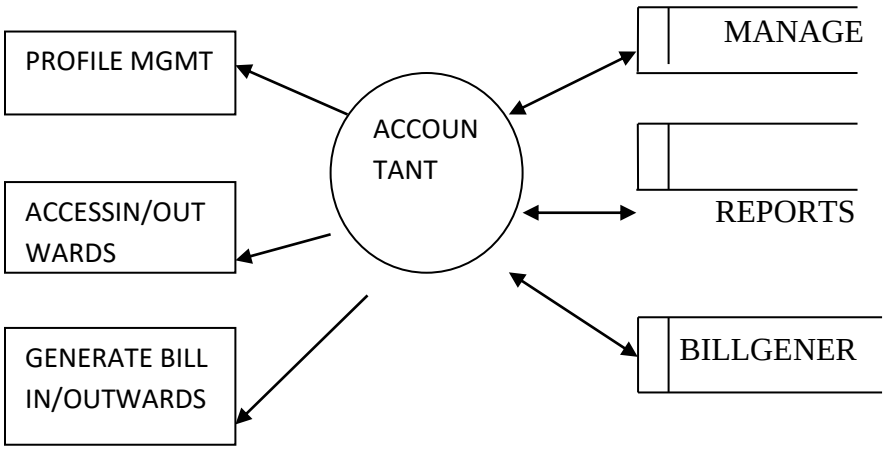
GR



ACCOUNTANT:DFD:

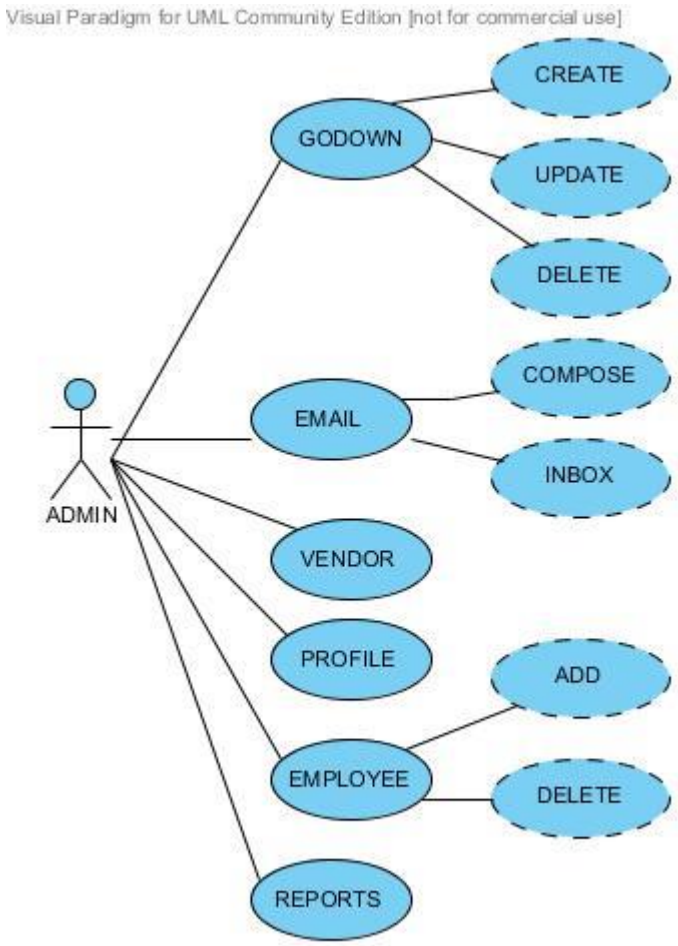
R

ATIO

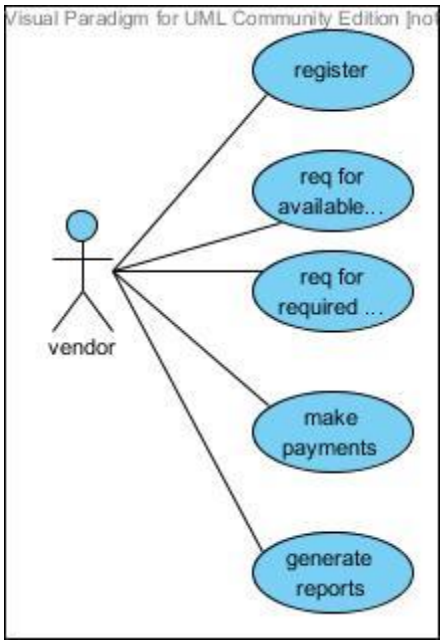


Usecase:

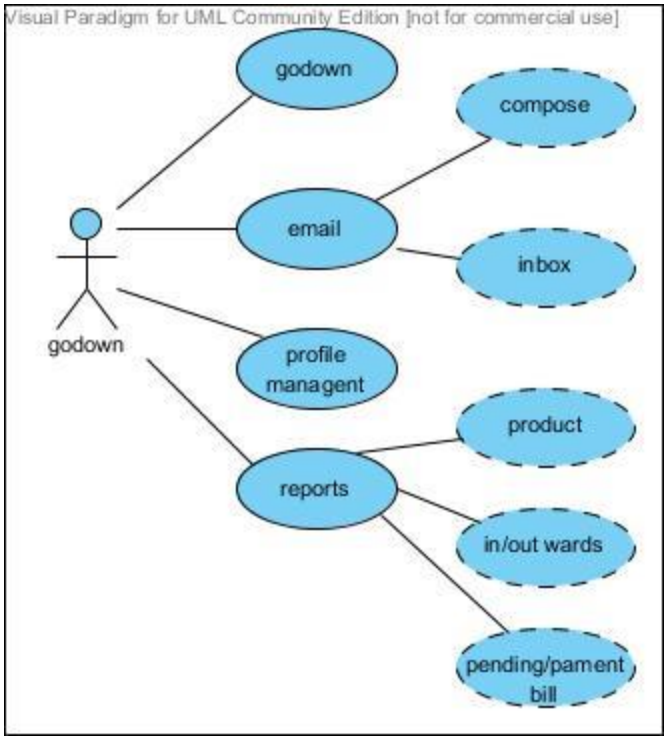
Admin:



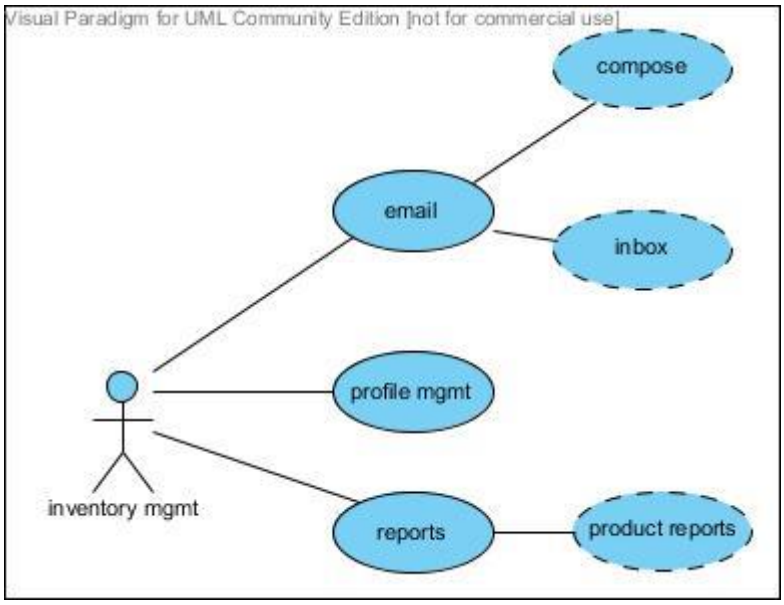
Vendor:


Visual Paradigm for UML Community Edition [not
register
req for available...
req for required ...
vendor
make payments
generate reports

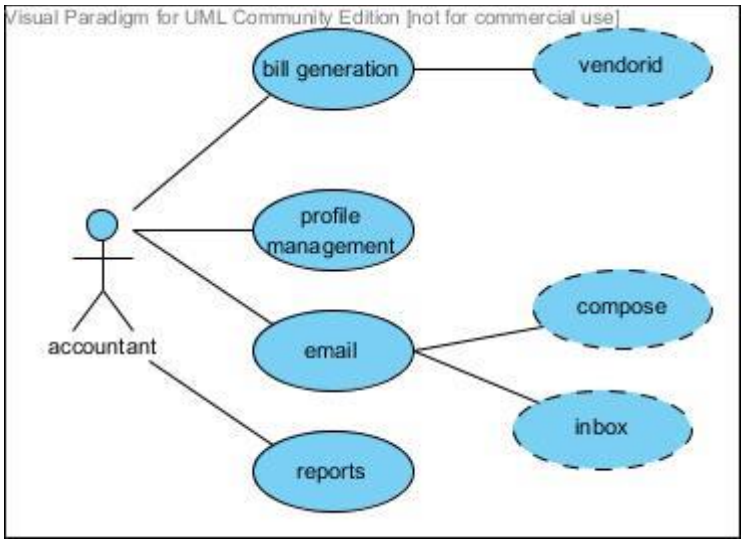
Godownmanager:



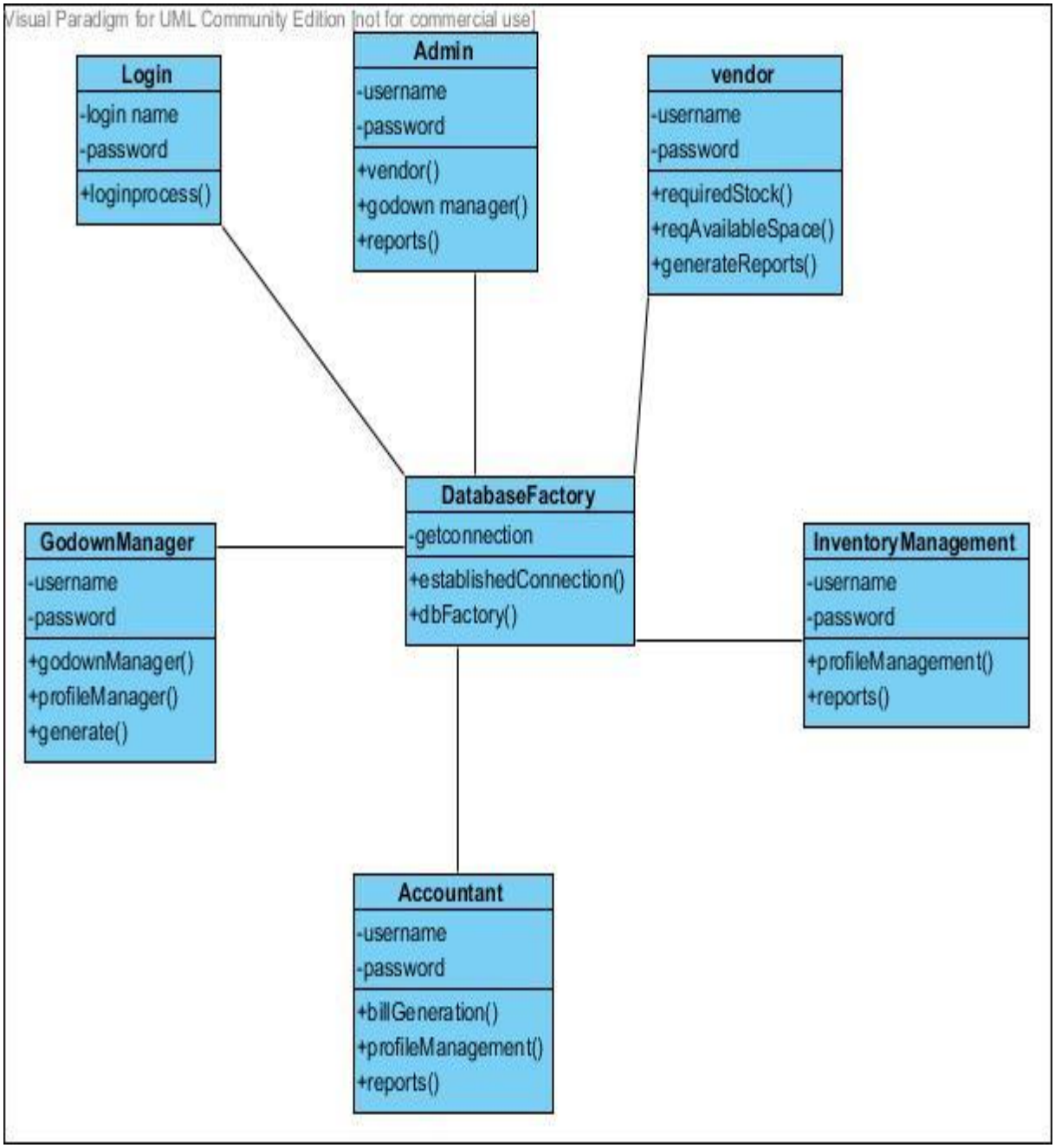
Inventory:



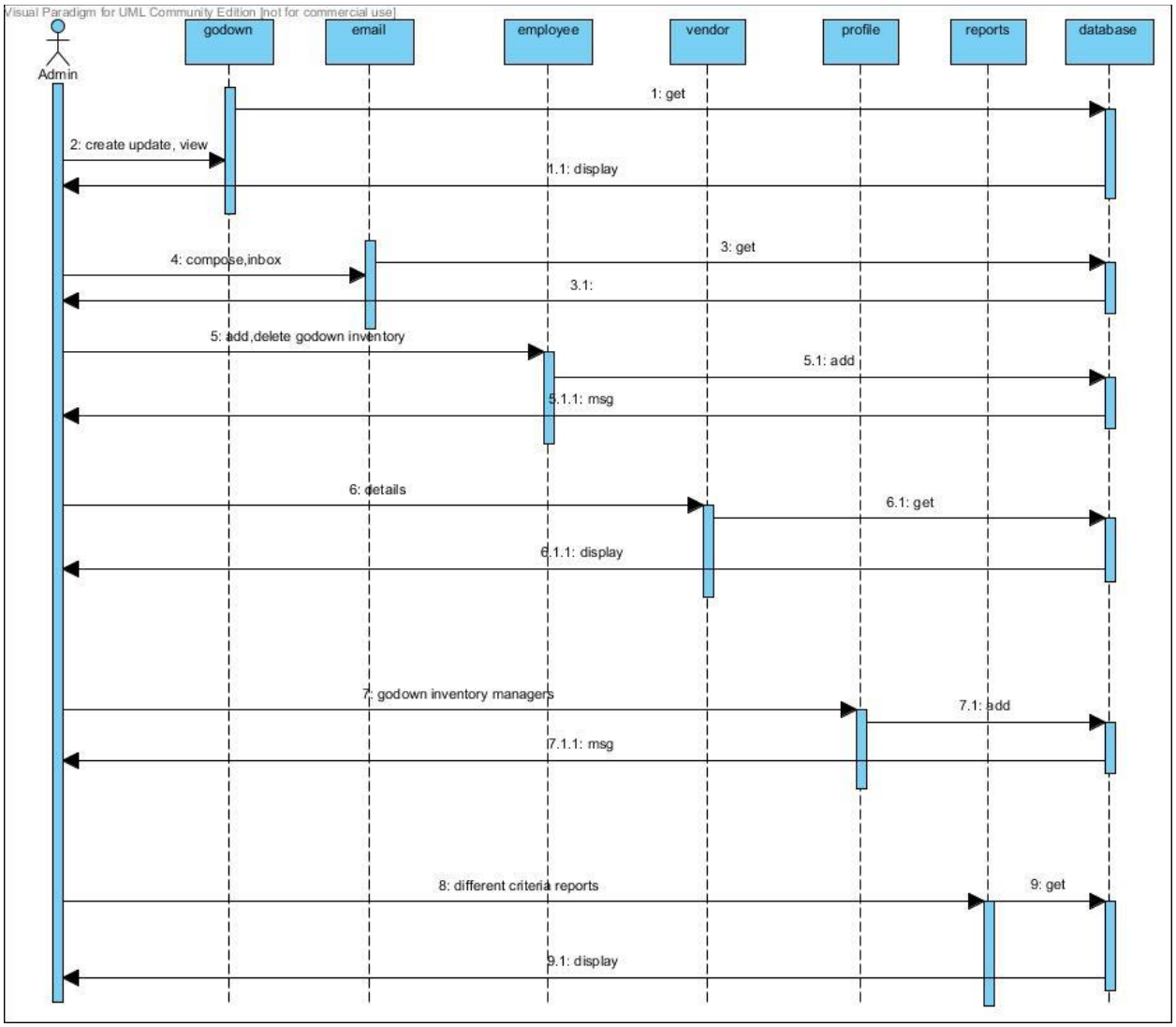
Accountant


Visual Paradigm for UML Community Edition [not for commercial use]
bill generation
vendorid
profile management
accountant
email
compose
inbox
reports

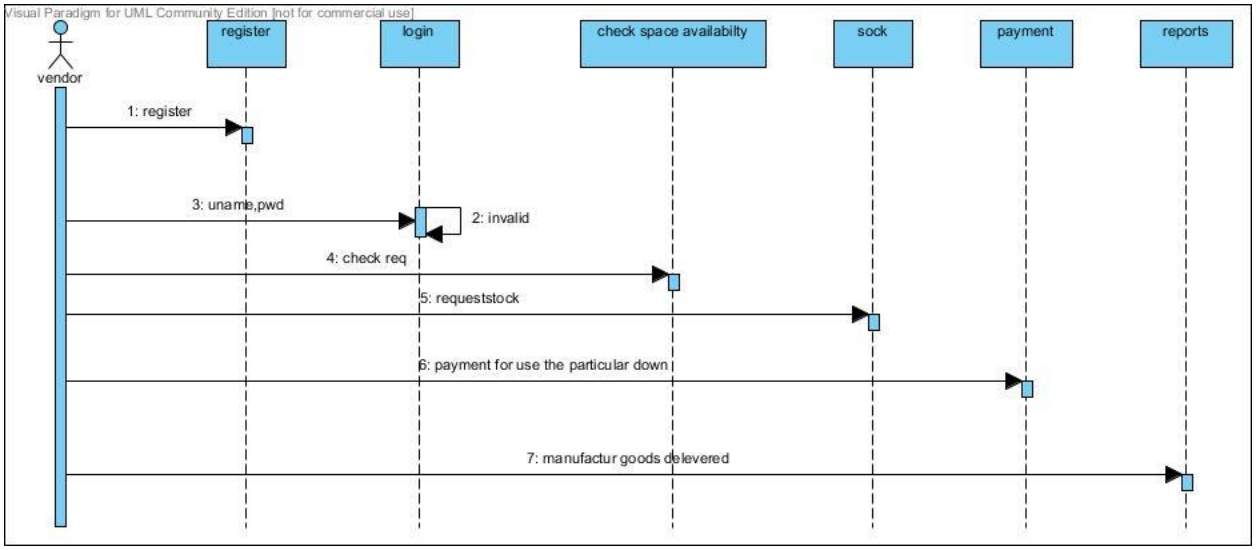
Class diagram:



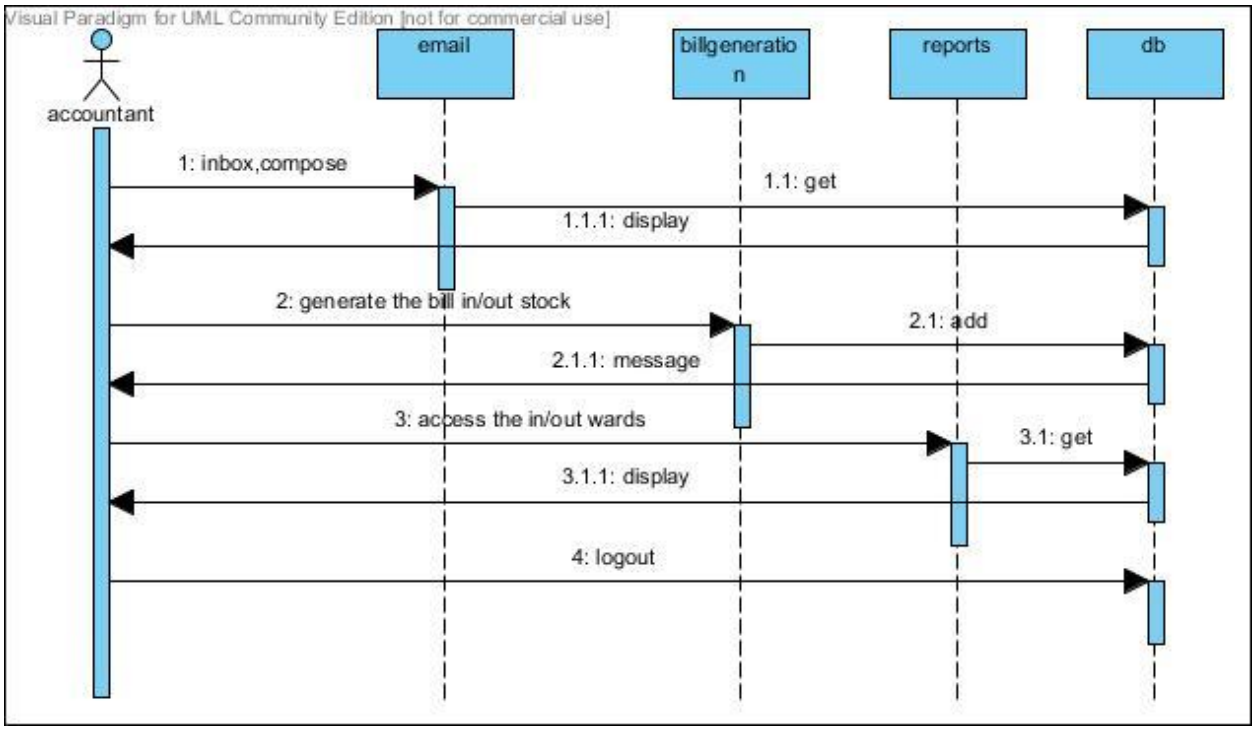
Admin


Visual Paradigm for UML Community Edition [not for commercial use]
Admin
godown
email
employee
vendor
profile
reports
database
1: get
2: create update, view
1.1: display
3: get
4: compose,inbox
3.1:
5: add,delete godown inventory
5.1: add
5.1.1: msg
6: details
6.1: get
6.1.1: display
7: godown inventory managers
7.1: add
7.1.1: msg
8: different criteria reports
9: get
9.1: display

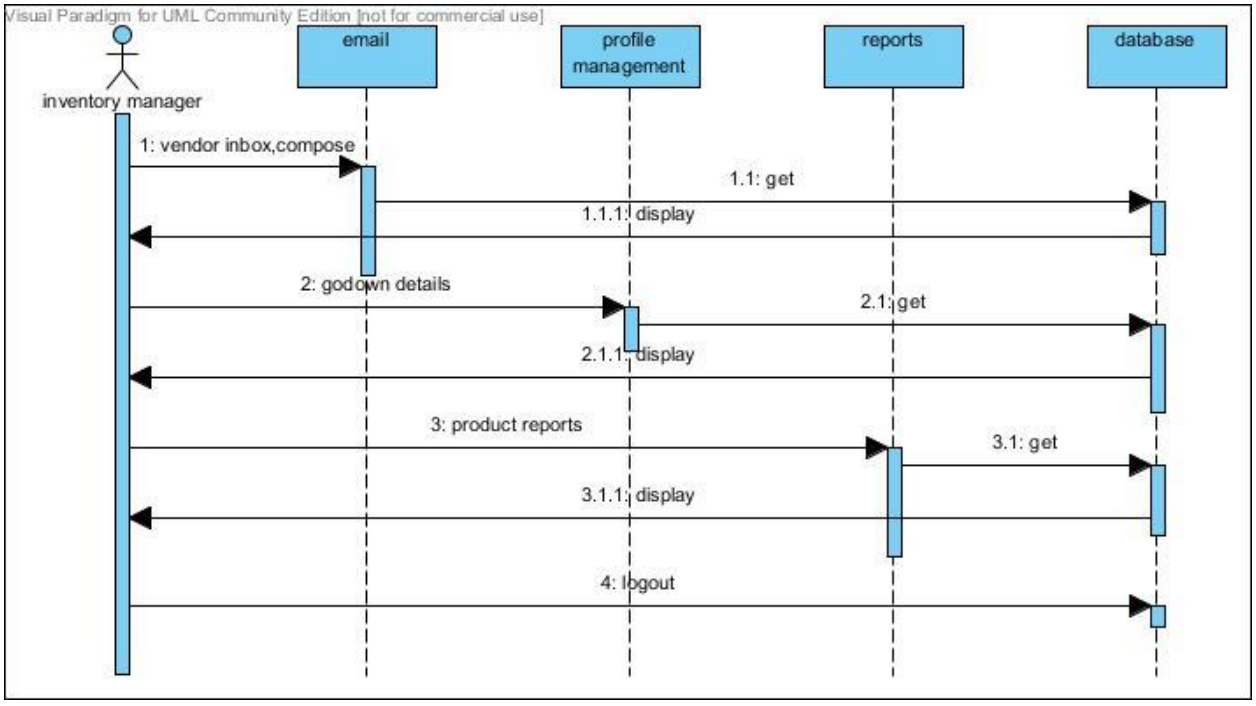
vendor


Visual Paradigm for UML Community Edition [not for commercial use]
vendor
register
login
check space availabilty
sock
payment
reports
1: register
3: uname,pwd
2: invalid
4: check req
5: requeststock
6: payment for use the particular down
7: manufactur goods delevered

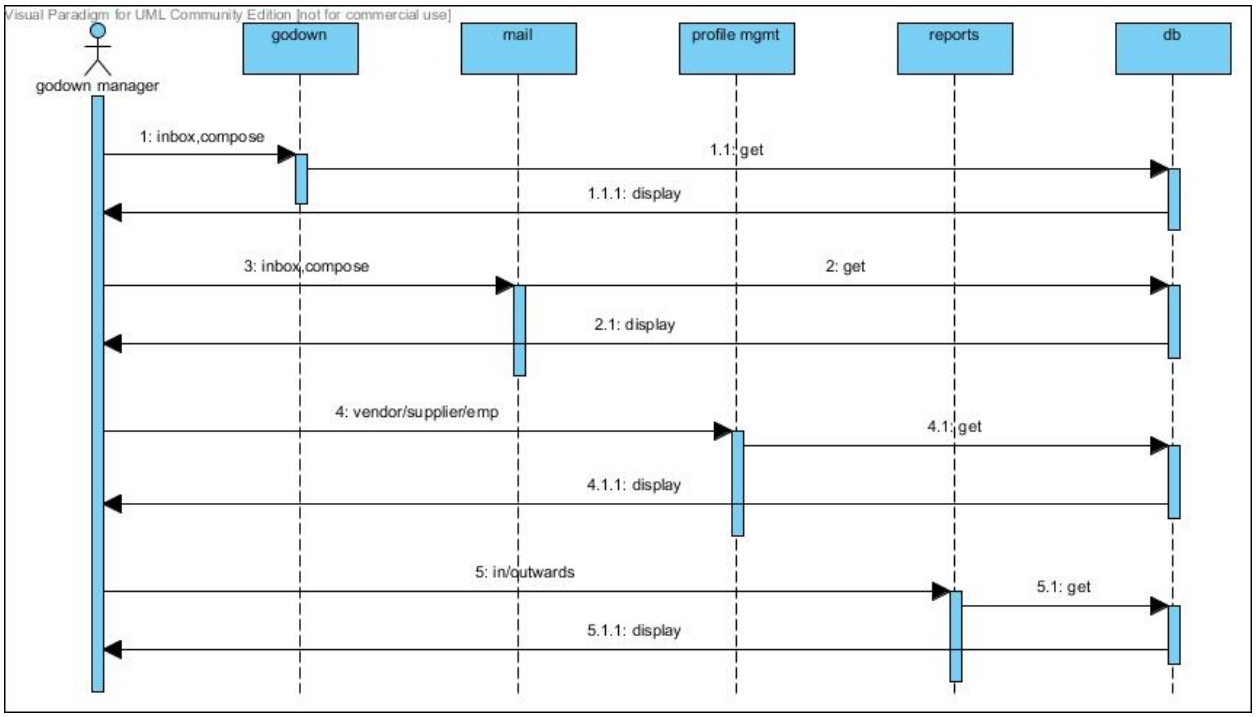
Accountant:


Visual Paradigm for UML Community Edition [not for commercial use]
accountant
email
billgeneratio
n
reports
db
1: inbox,compose
1.1: get
1.1.1: display
2: generate the bill in/out stock
2.1: add
2.1.1: message
3: access the in/out wards
3.1: get
3.1.1: display
4: logout

Inventory manager:

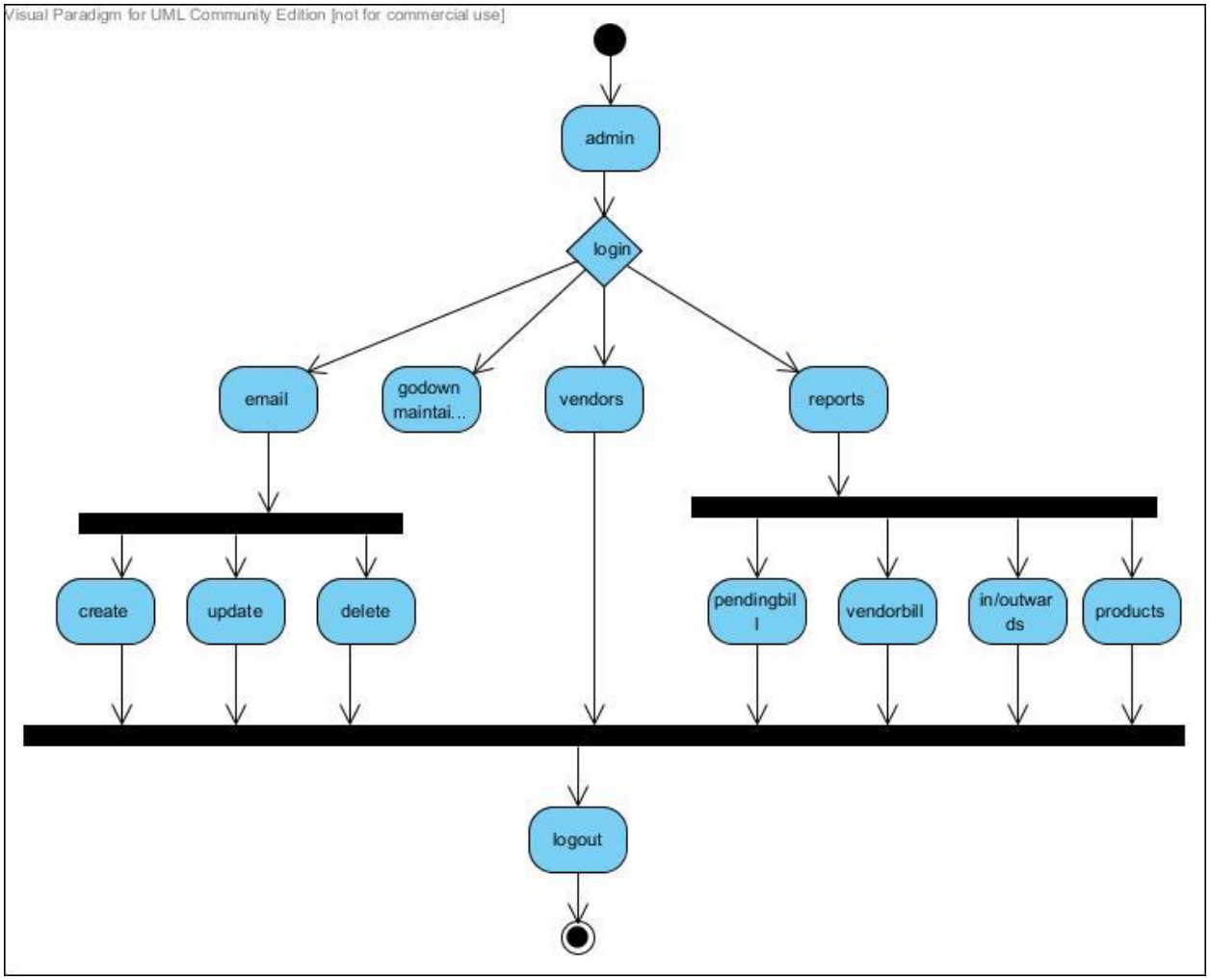


Godownmanager:

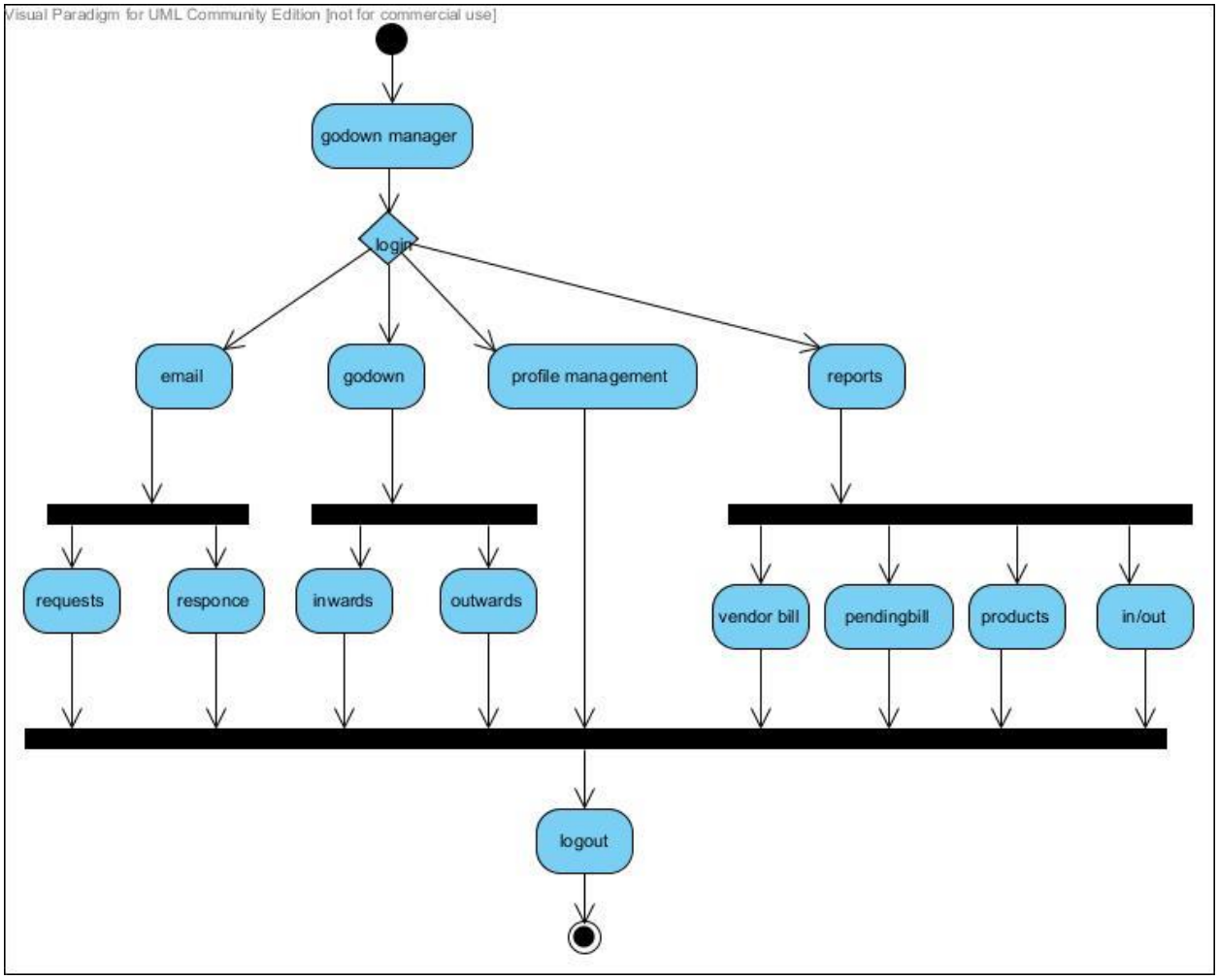


Activity:

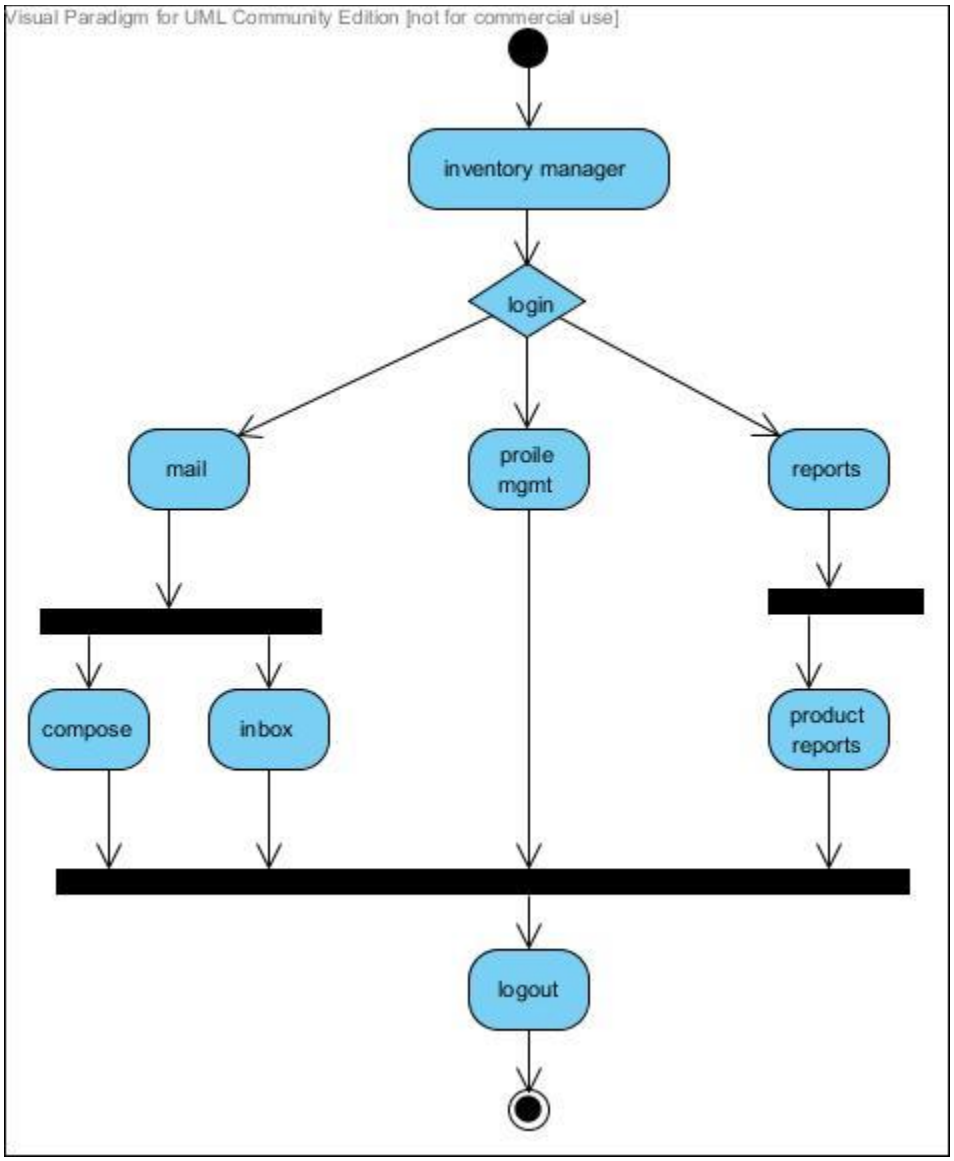
Admin:



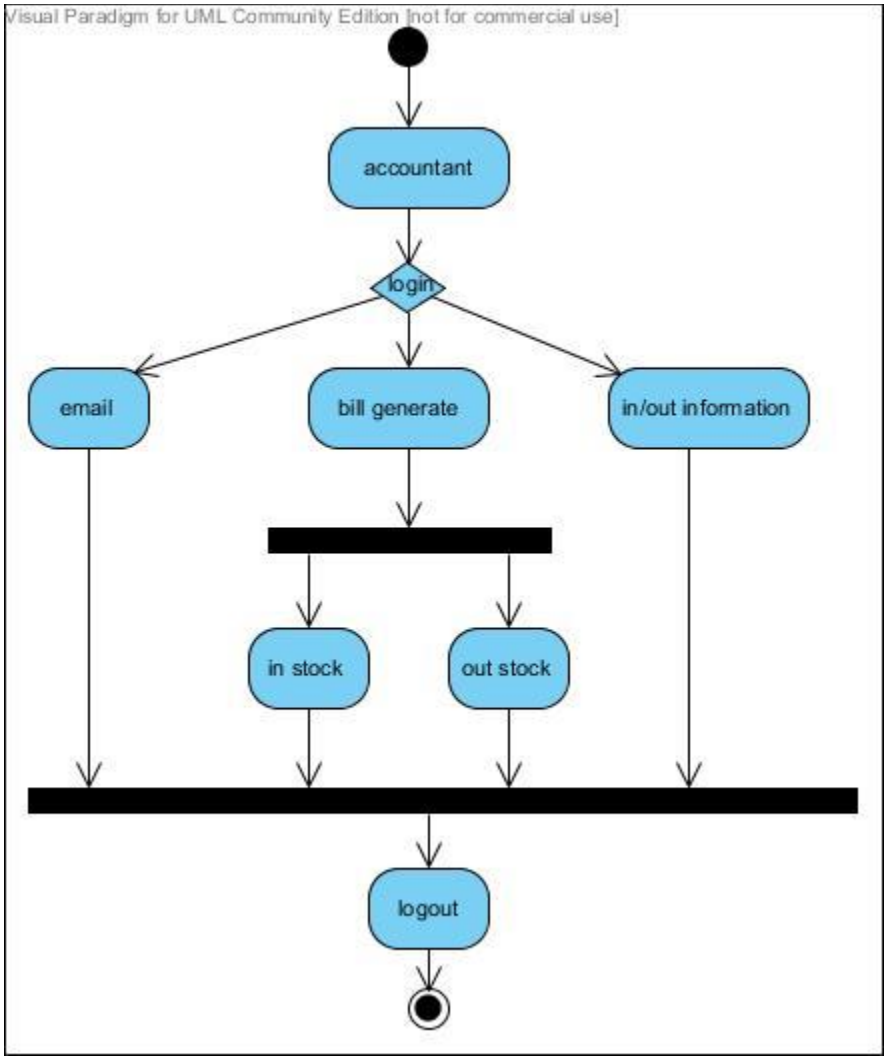
Godown manager:



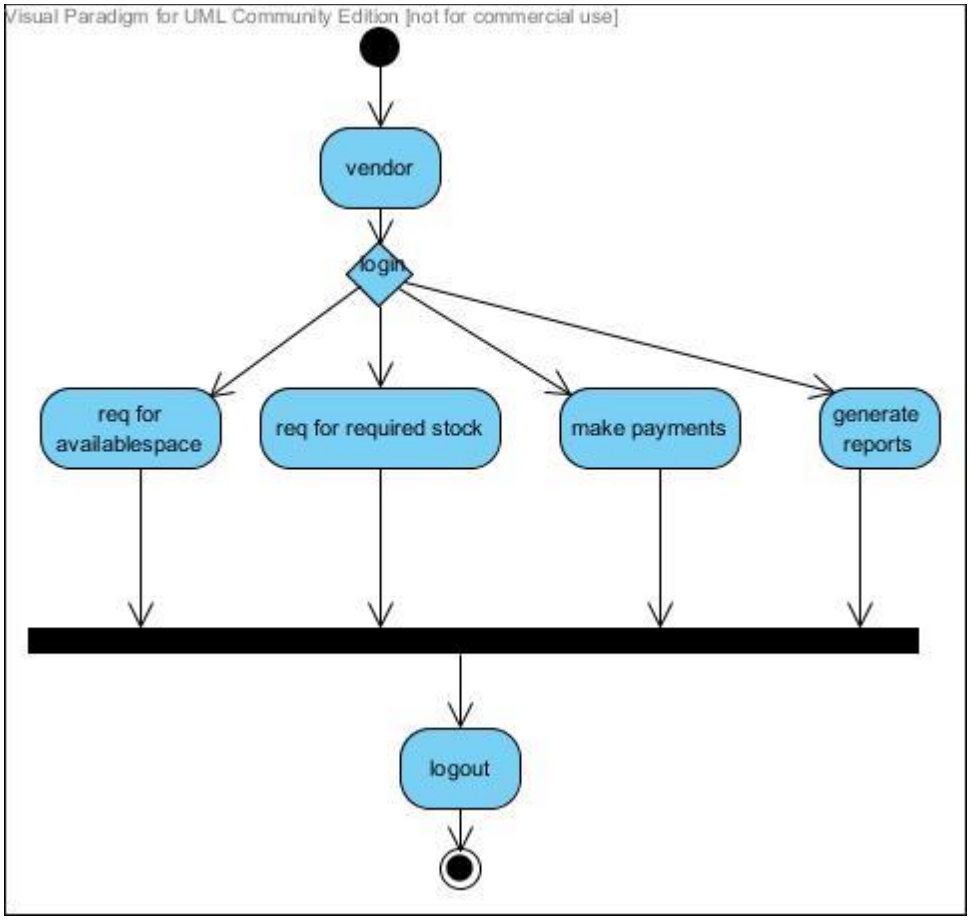
Inventory management:



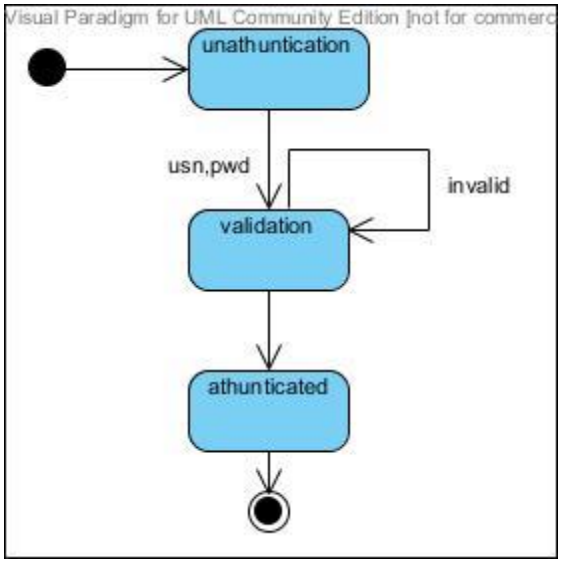
Accountant:

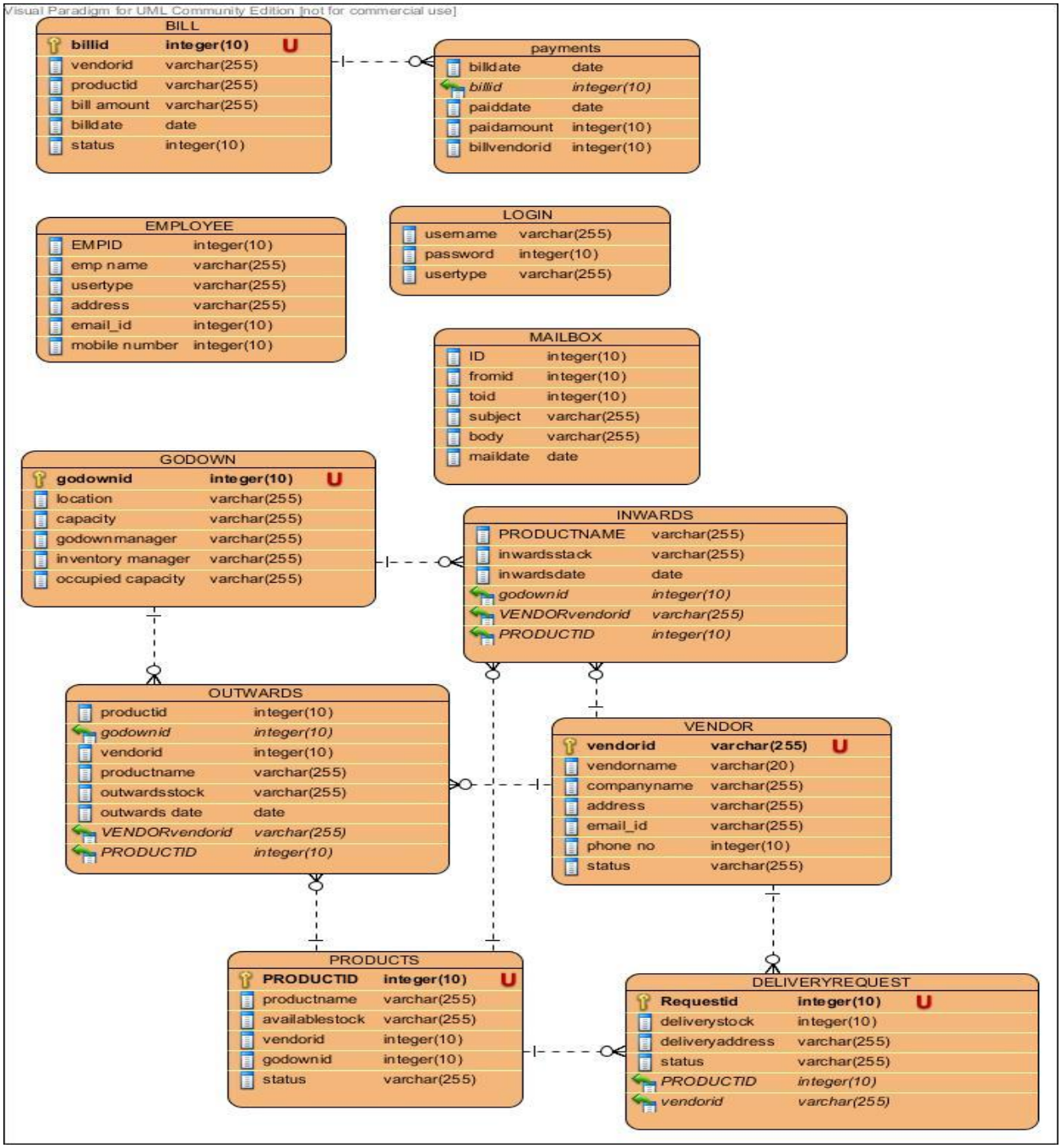

Visual Paradigm for UML Community Edition [not for commercial use]
accountant
login
email
bill generate
in/out information
in stock
out stock
logout

Vendor:


Visual Paradigm for UML Community Edition [not for commercial use]
vendor
login
req for availablespace
req for required stock
make payments
generate reports
logout

STATE CHART:


Visual Paradigm for UML Community Edition [not for commerc
unathuntication
usn,pwd
invalid
validation
athunticated


Visual Paradigm for UML Community Edition [not for commercial use]
BILL
billid integer(10) U
vendorid varchar(255)
productid varchar(255)
bill amount varchar(255)
billdate date
status integer(10)
payments
billdate date
billid integer(10)
paiddate date
paidamount integer(10)
billvendorid integer(10)
EMPLOYEE
EMPID integer(10)
emp name varchar(255)
usertype varchar(255)
address varchar(255)
email_id integer(10)
mobile number integer(10)
LOGIN
username varchar(255)
password integer(10)
usertype varchar(255)
MAILBOX
ID integer(10)
fromid integer(10)
toid integer(10)
subject varchar(255)
body varchar(255)
maildate date
GODOWN
godownid integer(10) U
location varchar(255)
capacity varchar(255)
godownmanager varchar(255)
inventory manager varchar(255)
occupied capacity varchar(255)
INWARDS
PRODUCTNAME varchar(255)
inwardsstack varchar(255)
inwardsdate date
godownid integer(10)
VENDORvendorid varchar(255)
PRODUCTID integer(10)
OUTWARDS
productid integer(10)
godownid integer(10)
vendorid integer(10)
productname varchar(255)
outwardsstock varchar(255)
outwards date date
VENDORvendorid varchar(255)
PRODUCTID integer(10)
VENDOR
vendorid varchar(255) U
vendorname varchar(20)
companyname varchar(255)
address varchar(255)
email_id varchar(255)
phone no integer(10)
status varchar(255)
PRODUCTS
PRODUCTID integer(10) U
productname varchar(255)
availablestock varchar(255)
vendorid integer(10)
godownid integer(10)
status varchar(255)
DELIVERYREQUEST
Requestid integer(10) U
deliverystock integer(10)
deliveryaddress varchar(255)
status varchar(255)
PRODUCTID integer(10)
vendorid varchar(255)

6. IMPLEMENTATION CODING

```
from django.db import models
class Admin(models.Model):
    username=models.CharField(max_length=30)
    password=models.CharField(max_length=30)

class Employee(models.Model):
    idno = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=30)
    branch = models.CharField(max_length=20)
    email = models.EmailField(max_length=100)
    contactno = models.IntegerField()
    address = models.CharField(max_length=100)
    city = models.CharField(max_length=20)
    categoryid = models.CharField(max_length=30)
    password = models.CharField(max_length=30)

class Categories(models.Model):
    categoriesID=models.IntegerField(primary_key=True)
    categoriesNAME=models.CharField(max_length=50)
    description=models.CharField(max_length=100)


class Product(models.Model):
    productId=models.IntegerField(primary_key=True
```

```
  productName=models.CharField(max_length=30)
    productCategory=models.CharField(max_length=30)
    productDescription=models.CharField(max_length=100)
    owner=models.CharField(max_length=30)
    phoneNo=models.IntegerField()
    emailId=models.EmailField(max_length=100)
    postDate=models.DateField()
    cost=models.DecimalField(max_digits=20,decimal_places=2)
    image=models.FileField(upload_to="media/image",default=False)

class Userlogn(models.Model):
    userId=models.IntegerField(primary_key=True)
    name=models.CharField(max_length=30)
    email_Id=models.EmailField(max_length=100)
    phone_No=models.IntegerField()
    address=models.CharField(max_length=100)
    password=models.CharField(max_length=150)
from django.shortcuts import render
from .models import Admin,Employee,Categories,Userlogn
from django.views.generic import CreateView,UpdateView,DeleteView
def openIndex(request):
    qs=Categories.objects.all()
    return render(request,"index.html",{"data":qs})

def adminLogin(request):
```

```
    username= request.POST.get("username")
    password=request.POST.get("password")
    # a=Admin(username=username,password=password)
    # a.save()
    qs=Admin.objects.filter(username=username,password=password)
    if qs:
        return render(request,"adminHome.html",)
    else:
        return render(request,"admin.html",{"msg":"Invalid Admin"})


class CreateEmp(CreateView):
    template_name = "createemp.html"
    model = Employee
    fields = ('idno','name','branch','email','contactno','address','city','categoryid','password')
    success_url = "/viewemp/"


class UpdateEmployee(UpdateView):
    template_name = "updateemp.html"
    model = Employee
    fields = ('name','branch','email','contactno','address','city','categoryid','password')
    success_url = '/viewemp/'


class DeleteEmployee(DeleteView):
    model = Employee
    success_url = '/viewemp/'
```

```
class AddCategories(CreateView):
    template_name = "addcategories.html"

    model = Categories
    fields = ('categoriesID','categoriesNAME','description')
    success_url = "/viewcategories/"

class UpdateCategories(UpdateView):
    template_name = "updatecategories.html"
    model = Categories
    fields = ('categoriesNAME','description')
    success_url = "/viewcategories/"

class DeleteCategories(DeleteView):
    model = Categories
    success_url = "/viewcategories/"

def changePassword(request):
    idno=request.POST.get("idno")
    qs=Employee.objects.filter(idno=idno)
    return render(request,"changepass.html",{"data":qs})

def change(request):
    old = request.POST.get("oldpass")
```

```
    new = request.POST.get("newpass")
    confirm = request.POST.get("confirmpass")
    idno=request.POST.get("t1")
    name=request.POST.get("t2")
    branch=request.POST.get("t3")
    email=request.POST.get("t4")
    cno=request.POST.get("t5")
    address=request.POST.get("t6")
    city=request.POST.get("t7")
    cat=request.POST.get("t8")
    pas=request.POST.get("t9")
    if old == pas and new == confirm:

Employee(idno=idno,name=name,branch=branch,email=email,contactno=cno,address=address,c
ity=city,categoryid=cat,password=new).save()
        return render(request, "changepassword.html", {"msg": "Password Changed Successfully"})
    else:
        return render(request, "changepassword.html", {"error": "Password Does Not Match"})


def employeeLogin(request):
    idno=request.POST.get("idno")
    password=request.POST.get("password")
    qs=Employee.objects.filter(idno=idno,password=password)
    if not qs:
        return render(request,"emplog.html",{"msg":"Invalid User"})
    else:
```

```
        return render(request,"empHome.html")


def userLogin(request):
    idno=request.POST.get("idno")
    password=request.POST.get("password")
    qs=Userlogn.objects.filter(userId=idno,password=password)
    print(qs)
    if qs:
        return render(request, "userHome.html",{"data":qs})
    else:
        return render(request, "userlog.html", {"msg": "Invalid User"})
# Generated by Django 2.1.4 on 2019-02-04 11:24

from django.db import migrations



class Migration(migrations.Migration):

    dependencies = [
        ('app', '0001_initial'),
    ]

    operations = [
        migrations.DeleteModel(
            name='Employee',
```

7. SYSTEM TESTING

7.1 INTRODUCTION TO TESTING

Introduction to Testing:

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During software development. During testing, the program is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as it is expected to perform.

7.2 TESTING IN STRATEGIES

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

Unit Testing:

Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements.

Each module can be tested using the following two Strategies:

Black Box Testing:

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

- Incorrect or missing functions
- Interface errors

- Errors in data structure or external database access
- Performance errors
- Initialization and termination errors.

In this testing only the output is checked for correctness. The logical flow of the data is not checked.

White Box testing:

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been uses to generate the test cases in the following cases:

- ✓ Guarantee that all independent paths have been Executed.
- ✓ Execute all logical decisions on their true and false Sides.
- ✓ Execute all loops at their boundaries and within their operational bounds
- ✓ Execute internal data structures to ensure their validity.

Integrating Testing :

Integration testing ensures that software and subsystems work together a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.

System Testing :

Involves in-house testing of the entire system before delivery to the user. It's aim is to satisfy the user the system meets all requirements of the client's specifications.

Acceptance Testing :

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

Test Approach :

Testing can be done in two ways:

- Bottom up approach
- Top down approach

Bottom up Approach:

Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded with in the larger system. When bottom level modules are tested attention turns to those on the next level that use the lower level ones they are tested individually and then linked with the previously examined lower level modules.

Top down approach:

This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written. A stub is a module shell called by upper level module and that when reached properly will return a message to the calling module indicating that proper interaction occurred. No attempt is made to verify the correctness of the lower level module.

Validation:

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.

TESTING

Test case : **Login module**

Purpose: Login module is being tested here. The Application has login module for all the three (i.e) Admin module, faculty module, Student module.

Prereq: The login pass contains two labels, email id and password with the corresponding text fields.

Test Data: Providing Invalid email id for admin and testing is performed to check Admin

Homepage is accessed or not.

Steps: 1.Go to login Page

2. Enter no email id and password

3. Click on Submit button and checking the validations

4. Enter correct email id and password (i.e) admin and admin

5. Homepage is shown

6. verifying the homepage is Admin Homepage

Test Screens: The following screens represent the above test cases

8.SYSTEM SECURITY

8.1 INTRODUCTION:

Setting Up Authentication for Web Applications

Introduction:

To configure authentication for a Web Application, use the <login-config> element of the web.xml deployment descriptor. In this element you define the security realm containing the user credentials, the method of authentication, and the location of resources for authentication.

8.2 SECURITY IN SOFTWARE

To set up authentication for Web Applications:

1. Open the web.xml deployment descriptor in a text editor or use the Administration Console. Specify the authentication method using the <auth-method> element. The available options are:

   BASIC

   Basic authentication uses the Web Browser to display a username/password dialog box. This username and password is authenticated against the realm.

   FORM

   Form-based authentication requires that you return an HTML form containing the username and password. The fields returned from the form elements must be: j_username and j_password, and the action attribute must be j_security_check. Here is an example of the HTML coding for using FORM authentication:

   <form method="POST" action="j_security_check">

   <input type="text" name="j_username">
   <input type="password" name="j_password">

</form>

The resource used to generate the HTML form may be an HTML page, a JSP, or a servlet. You define this resource with the <form-login-page> element.

The HTTP session object is created when the login page is served. Therefore, the session.isNew() method returns FALSE when called from pages served after successful authentication.

9. BIBLIOGRAPHY

HTML

HTML Black Book by Holzner

PYTHON

SQL LITE Database Programming with DJANGO SERVER by Patel moss.

Software Engineering by Roger Pressman

SYSTEM STUDY

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user

expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.