

Software Engineering Final Assessment

Pharmacy Management System

**Program: Computer Engineering and Software Systems**

***Course Code:*** ***CSE334***

***Course Name: Software Engineering***

***Examination Committee***

**Dr. Gamal A. Ebrahim**

**Dr. Mohamed A. Taher**

**Dr. Mahmoud I. Khalil**

**Ain Shams University**

**Faculty of Engineering**

**Spring 2020 Semester**

Software Engineering

Final Assessment

Daniel Tarek ElShabrawy 18P1185

CESS

Faculty of Engineering, Ain Shams University

18P1185@eng.asu.edu.eg

Supervised by: Dr. Gamal Ebrahim

**Student Personal Information**

|  |  |
| --- | --- |
| **Student ID:** | 18P1185 |
| **Student Full Name:** | Daniel Tarek Lewis Saad Hanna ElShabrawy |

**Plagiarism Statement**

**Plagiarism Statement**

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Daniel Tarek Lewis Saad Hanna ElShabrawy

Date:

30/5/2020

Signature/Student Name:

**Submission Contents**

Contents

[I-Introduction 6](#_Toc41720028)

[1.0 Software Development Process Models 7](#_Toc41720029)

[1.1 Waterfall Process Model 7](#_Toc41720030)

[1.2 Incremental Model 7](#_Toc41720031)

[1.3 Spiral Model 8](#_Toc41720032)

[1.4 Agile Process Model 8](#_Toc41720033)

[1.5 V-Shaped Model 8](#_Toc41720034)

[2.0 Software project description. And an application of each phase of these models on it. 9](#_Toc41720035)

[2.1 Software Project and Description 9](#_Toc41720036)

[2.1.1 Manual Pharmacy System Problems. 9](#_Toc41720037)

[2.1.2 How will the Pharmacy Management System affect? 9](#_Toc41720038)

[2.1.3 AIM AND PURPOSE OF THE PROJECT 9](#_Toc41720039)

[2.2 Applying Each Phase of the Process Models 10](#_Toc41720040)

[2.2.1 Waterfall Model 10](#_Toc41720041)

[2.2.2 Incremental Model 11](#_Toc41720042)

[2.2.3 Spiral Model 11](#_Toc41720043)

[2.2.4 Agile Process Model 11](#_Toc41720044)

[2.2.5 V-Shaped Model 12](#_Toc41720045)

[3.0 Cost Estimation 14](#_Toc41720046)

[3.1 COCOMO II Model 14](#_Toc41720047)

[3.2 Line of Code Model 15](#_Toc41720048)

[3.3 Functional Points Model 16](#_Toc41720049)

[4.0 Software Architecture 18](#_Toc41720050)

[4.1 Object-Oriented Architecture 18](#_Toc41720051)

[4.1.1 What is Object-Oriented Architecture 18](#_Toc41720052)

[4.1.2 Noun Extraction and CRC Cards 19](#_Toc41720053)

[4.1.3 Pharmacy Management System UML Class Diagram 20](#_Toc41720054)

[4.1.4 Use Case Diagram 21](#_Toc41720055)

[4.1.5 Use Case Description 22](#_Toc41720056)

[4.1.6 UML – State Diagram 29](#_Toc41720057)

[4.1.7 UML – Sequence Diagram 30](#_Toc41720058)

[4.2 Layered Architecture 31](#_Toc41720059)

[4.2.1 Data Flow Diagram - DFD Context 31](#_Toc41720060)

[4.2.2 DFD – Level 0 32](#_Toc41720061)

[4.2.3 DFD – Level 1 33](#_Toc41720062)

[4.2.4 Component Diagram 34](#_Toc41720063)

[4.3 Merged Styles Architecture 35](#_Toc41720064)

[4.4 Cohesion and Coupling 36](#_Toc41720065)

[4.4.1 Coupling. 36](#_Toc41720066)

[4.4.2 Cohesion 37](#_Toc41720067)

[5.0 GUI 38](#_Toc41720068)

[6.0 References 40](#_Toc41720069)

# I-Introduction

The following report is a simple yet very important Software System based on a research from many different sources This Pharmacy Management System may help many pharmacies manage their stocks and use technology to account for profits, taxes and other important record without explicitly hiring a professional to do so. This report contains the basic yet fundamental information and diagrams required to develop this Pharmacy Management System including the UML diagrams and Data Flow Diagrams. The system basically takes an order from a customer and records it on a database to be sent to the delivery employee and sent to the supplier to order a replacement of the inventory sold. Also keeps records of the most demanded medicines and reports it to the user. I hope that this software helps pharmacy stores become more organized when it comes to stock and records management.

# 1.0 Software Development Process Models

A Process Model defines the sequence of phases over a product's entire lifetime. Hence, it may also be called Product Life Cycle. This covers everything from the initial commercial concept right up to the final reinstallation or disassembly after use of the product. SDLC is a procedure practiced within the software organization, for a software project. The development of the program should be complete within the predefined time and cost. It consists of a detailed plan, which describes how specific software can be created, maintained, replaced and updated. Each phase of the Software Development Life Cycle has its own processes that delivers it to the next phase. The SDLC ensures that from start to finish, you do not miss a step. Today various SDLC methodologies are used to direct professionals through their project based work. Below are the main pros and cons of six of the most common methodologies used by the SDLC.

## 1.1 Waterfall Process Model

Waterfall is the oldest and simplest of the standardized SDLC methodologies — finish a step, and then move on to the next. No return. Each stage is based on previous stage details, and has its own project plan. Waterfall is quick to understand and easy to manage. Early delays can however throw off the entire timetable of the project. Moreover, since there is no space for changes once a process is complete, issues cannot be resolved before you enter the maintenance process. The model does not fit well when there is a need for flexibility or if the project is ongoing and long term. Some of the Waterfall Process Model Advantages:

* Easily understood and used.
* Easily managed due to the rigidity of the model. Each phase is reviewed before entering the next one.
* Waterfall Process Model works better for relatively smaller projects where requirements are understood.

The waterfall model is based on techniques used in manufacturing and other physical industries. Basically it relies on a sequential behavior where each step is finished separately without overlapping the successive phase. This is essential when it comes to catching bugs before they become an issue as each phase is reviewed alone. You can also better estimate how long the entire SDLC will take, and set aside a portion of your schedule for each stage in the process, with the bulk of it on coding.

Winston W. Royce originally outlined the method in 1970, with six main phases:

* Requirements
* Analysis
* Design
* Coding
* Testing
* Operations

The waterfall model also calls for extensive documentation, making it easy for new team members to catch up on what happened before they arrived. The downside is that this is a more rigid approach, so it is harder to change course if the terms of your project change. It also does not provide the option of releasing an MVP, or minimum viable product, early in the research and development phase. As a result, some developers use a modified waterfall that allows for some overlapping phases, which we will look at later.

## 1.2 Incremental Model

Works by constructing a partial implementation of a total system and then slowly increase the functionality. The incremental model prioritizes device specifications and then implements them together in classes. Each subsequent program release adds function to the previous version, before all planned features have been implemented. This works on the basis that the entire process is split into several iterations, which is nothing but the partial implementation of a total process. Then new modules are added in these iterations, which keep increasing their functionality. The development process can be either sequential or parallel. The incremental differs from the Waterfall; instead of taking a sequential approach, this is more flexible. However, waterfall may be used inside the incremental approach in smaller stages usually referred to as ‘mini waterfalls.‘ This allows stages to add new ideas or functionalities that were not initially in the starting phase needless to start all over. Multiple iterations may be done simultaneously referred to as parallel development. Using interchangeable parts or modules, as each cycle can build on the next; this approach can result in a shorter development timeframe. The limitations to this approach include more input required from the customer or client, and a reduced capacity to handle larger, non-iterative components.

## 1.3 Spiral Model

In this approach both concept elements and prototyping components. In spiral model, testing is performed at nearly every software phase / module, rather than testing the entire program immediately after its development. This approach adds to the software development progress as it is checked after each step for the desired performance. Additionally, if found, it is easier to go back to the previous stage in the spiral model after checking for any changes. Not to mention that the prototyping principle contributes to the creation process. Finally, the Spiral SDLC is a convergence of all of the listed SDLC methods, because it enables multiple methodologies to be combined into a single development project. Teams are allowed to switch between types of SDLCs based on project risk rates. The technology proved successful for projects with high levels of ambiguity and variation in requirements. As you can see from the summary presented, today there are ranges of SDLC forms to help tech firms minimize the risk of their technology, save time and money, strengthen investment decisions and make use of the best business practices to streamline organizational processes.

## 1.4 Agile Process Model

Other SDLC models fall under this category as these models pay less attention to comprehensive software documentation and more attention to software testing. This allows quicker distribution but makes their management more difficult due to the lack of thorough explanation of the program. Nowadays, due to the availability of iterative growth, intense communication and early customer input, most businesses employ this group's models for their IT ventures.

* Scrum

Scrum is the most popular agile model. Here the iterations are termed as ‘Sprints’ and are usually 2-4 weeks long and they are preceded with thorough planning and sprint management. No changes are further allowed after the sprint activities are defined.

* XP

XP stands for Extreme Programming and it generally lasts for about 1-2 weeks. This model is flexible even after the iteration’s launch however; this causes complications in the delivery of quality software. To avoid the problem, XP uses the concept of pair programming, test-driven development and test automation, continuous integration (CI), small releases, simple software design and following of several coding standards.

* Kanban

The distinguishing feature of this model is the absence of pronounced iterations. The model uses extremely short or ‘Daily Sprints’ and an emphasis is placed on plan visualization. Therefore, the team tends to use the Kanban Board tool, which provides a clear representation of all project activities, their number, responsible personnel, and progress. Due to this, there is increased transparency in this model. In addition, there is no separate planning stage, so a new change request can be introduced at any time. Kanban model is generally used for projects on software support and evolution.

## 1.5 V-Shaped Model

This model is a Waterfall extension, with an emphasis on the study. It works well for projects requiring significant control and clearly defined and unchanging specifications. Small projects are typically too heavy-handed, so changes in specifications can result in lengthy processes so delays. It is a type of linear model where there is a corresponding test operation at each point. It is quite similar to the Waterfall model, where in the beginning all requirements are gathered and cannot be changed later. Mistakes and errors can be identified early via this model but the implementation of adjustments during production is often costly and hard to enforce. Due to the workflow of this model, it is considered one of the most expensive and time consuming process models. V-model is generally used for projects where failures and downtimes are unacceptable such as in medical, aviation domains software.

# 2.0 Software project description. And an application of each phase of these models on it.

## 2.1 Software Project and Description

Technology nowadays plays a significant role within the healthcare system. A technologically appropriate, equitable, affordable and efficient and consumer friendly system, designed to be used to the maximum benefit of the pharmaceutical industry must be ensured. A Pharmacy Management System is being developed to leverage the Computer Technology principles. The Pharmacy Management System is a robust, integrated technology that handles drug and consumables maintenance in the pharmacy units. This software system will ensure sufficient amount of drugs are available. The key objective of the system is to automate the current system of manually managed counter sales records, orders, reorder rates, monetary positions of suppliers and customers and other relevant seller transactions. Any other store may use this program to automate the process of manually keeping records related to the subject of holding stock and fluid flows.

### 2.1.1 Manual Pharmacy System Problems.

Most pharmacies still do their entire work manually; this manual system requires pharmacists or workers to manually monitor the entire process and check each drug's presence in Pharmacy. And when the latest products or new product supplies arrive at the Pharmacy, the manual entry in the register is completed, and this followed when administering the drug to any patient. When the month is over the pharmacy staff have to manually produce the list or record the drugs in the pharmacy. This research is being done to maintain the Pharmacy's need stock. This form of work could lead to mistakes. The pharmacy management maintained records of paper in cabinet filing. Managing a very large pharmacy with paper records would be cumbersome and difficult to keep track of stocks in store about medications, expiry date, quantity of drugs available depending on the categories and their functions. To replenish the already dwindling stock, the pharmacist must order medications. Additionally, medication ordering is performed manually. Significant amount of time is devoted to order writing, because the pharmacist has to go through the stock balance and make a rough calculation of the amount to order based on that. Drugs cannot be used until they have expired. This project research would prompt the pharmacist on medications that are close to expiry, avoiding the selling of such medications and thus offering a solution to the previously mentioned problems.

### *2.1.2 How will the Pharmacy Management System affect?*

To overcome these issues, there is an urgent need to establish a pharmacy management program that will be helpful to the pharmacy. Through using this app, we can generate bills, hold stock very well, save costs and maintain control over inventories. This program will support pharmacy; more easily and better manage the incomes and outgoings.

### 2.1.3 AIM AND PURPOSE OF THE PROJECT

* + Provide for mass storage of relevant data.
  + Make access to the data easy for the user.
  + Provide prompt response to user requests for data.
  + Making modifications to the database available immediately.
  + Allow multiple users to be active at one time.
  + Protect the data from unwanted and unauthorized access.
  + Get alert bout the medicines that are close to become expired.
  + Get Alert about the medicine low in stock or almost finished.
  + Have a good statistics part to know how much profit gained.

## 2.2 Applying Each Phase of the Process Models

### 2.2.1 Waterfall Model

As mentioned above the waterfall process model consists of five main phases.

2.2.1.1 The Requirements Phase. To start with, the first phase is the requirements phase. During this phase, all possible system specifications to be created are collected and recorded in a Requirements Specification Document. In relation to the Pharmacy Management System, the requirements phase will contain both functional and non-functional requirements of the software. The functional requirement is to explain the system behavior as it pertains to the functionality of the system. The non-functional requirement creates a device characteristic of efficiency. In relation to the Pharmacy Management System project, functional requirements might include the following functionalities:

* Add a medicine to storage.
* Remove a medicine from storage
* Calculate revenue, cost and profit
* Record the most and least demanded drugs
* Automatically order a medicine when it is low in stock
* Offer discounts to frequent customers
* Calculate tax
* Update price

As for the non-functional requirements, it may include the speed at which the Pharmacy Management System operates at or the level of security of the database, the time required to develop and test the software. Reliability is also important, to make sure that the system does not crash on a busy day.

2.2.1.2 The Design Phase. After the requirements phase, comes the design phase, this phase examines how the software will be built, and how the system will operate with special emphasis on hardware, software, network infrastructure, and user interface. The main aim of this stage is to establish a model that meets all recorded specifications, then define all relevant inputs, processes and outputs and help avoid misunderstandings by involving stakeholders including managers and users. There are two architecture types to remember. Firstly, the conceptual design that involves the design of the forms and reports, the system design and the design of the database. Then we have the actual architecture of the actual database of the Pharmacy, which consists of the pharmacists and users, the programs and processes and the distributed systems. We have a system-design specification document (SDS) at the end of the design process. This is a paper containing all the details needed to create the program. This is targeted at the involving implementation and testing teams as it provides expectations for what is required from the new program. To name a few thing which the SDS includes such as: a project scope, system design, component and process design, data handling, user displays and output reports, system files, and prototype analysis.

#### 2.2.1.3 The Implementation Phase.

This phase deals with the physical design of the software as set out in the design stage. Normally a production team consisting of programmers, system designers and other specialists accomplishes this. As well as designing practical and stable applications, this process also records research completed to date and offers assistance to current and planned users of the program. To relate to the Pharmacy Management System project, implementation of the basic functionalities of the system will be implemented using JAVA Language.

#### 2.2.1.4 Integration and Testing Phase

After testing each unit all the units built during the implementation process are incorporated into a system. To find and decide whether any bugs are present or not, the developed software needs to go through several and continuous tests. Monitoring is performed so that no problem faces the client during program deployment. The integration and testing phase in the Pharmacy Management System will include integrating the different functionalities of the system such as the Adding and removing of products, printing receipts and calculating profit together with the pharmacies database and hardware and the end-user tests the modules (alpha testing).

#### 2.2.1.5 Delivery and Maintenance

Its phase takes place after installation and involves making system or individual component modifications to alter attributes or improve performance. These changes emerge either from customer-initiated demands for improvement, or from defects found during the system's live use. Ongoing maintenance and support for the existing software is provided to the client. Different end-user test the modules to check for bugs (Beta testing).

### 2.2.2 Incremental Model

Incremental Model is a software development model where the specifications of the software development cycle are divided into several standalone modules. Each module in this model goes through the phases of requirements, design, implementation, and testing. Every subsequent module release adds feature to the preceding release. The process continues until the complete system reaches completion.

#### **2.2.2.1 Requirement analysis: The product design experience defines the criteria in the initial step of the incremental model. And the appropriate engineering team understands the device functional requirements. This phase performs a crucial role in developing the software under the incremental model. Similar to the waterfall model, the requirements analysis of the Pharmacy Managements System will include functionalities such as adding and removing medicine from storage, calculating profit, revenue and cost. The requirements may also be non-functional such as System Security, Response time etc.**

**2.2.2.2. Design & Development:In this phase of SDLC's Incremental Model, the system functionality design and the method of development are completed with success. The incremental model utilizes style and creation process as software creates new practicality. Also similar to the waterfall model,** implementation of the basic functionalities of the system will be implemented using JAVA Language.

#### **2.2.2.3.** **Testing: The testing process tests the output of each current feature and new functionality within the incremental model. During the testing process, the different methods are used to check each task's actions.** The integration and testing phase in the Pharmacy Management System will include integrating the different functionalities of the system such as the Adding and removing of products, printing receipts and calculating profit together with the pharmacies database and hardware and the end-user tests the modules (alpha testing).

#### 2.2.2.4. Implementation: Implementation phase allows for the software system coding process. In the design and development process, it includes the final coding the design and checks the software in the test phase. After this process is complete, the number of the product working is increased and upgraded to the final device product.

### Spiral Model

Every spiral loop is called a phase of Software Creation. The exact number of steps required to produce the product will vary by project manager depending on the project risks. Since the project manager dynamically determines the number of steps, the project manager plays an important role in the design of a product using the spiral model. The Spiral Model rising phase is split into four quadrants. The roles of the four quadrants are described below:

* + - 1. Objectives determination and identify alternative solutions:  Consumers get requirements and the requirements are specified, elaborated and evaluated at the beginning of each process. Then, alternative possible solutions for the process are provided within this quadrant.
      2. Identify and resolve Risks: All alternative solutions are processed to select the best possible solution, during the second quadrant. The risks related to the solution are then identified, and the risks are handled using the best possible strategy. At the end of this quadrant, prototype is designed to get the best possible solution.
      3. Develop next version of the Product: The identified features are then established and tested, during the third quadrant. By the end of the third quadrant, the successive version of the Software is available.
      4. Review and plan for the next Phase: The Customers test the version so far built in the fourth quadrant of the program. In the end, planning for the next step is begun.

### 2.2.4 Agile Process Model

Within the Agile development model, each iteration is considered a short "frame" period, usually lasting from one to four weeks. Dividing the whole project into smaller pieces, helps mitigate project risk and reduce the overall criteria for project completion times. Through iteration requires a team going through a full life cycle of software development including preparation, review of specifications, design, coding, and testing before a work product is presented to the client.

2.2.4.1. Requirements gathering: You need to define the requirements in this process. You will clarify market opportunities and schedule the time and energy necessary for the project to be completed. Based on that knowledge, technological and economic feasibility can be evaluated. **Similar to the waterfall and incremental model, the requirements analysis of the Pharmacy Managements System will include functionalities such as adding and removing medicine from storage, calculating profit, revenue and cost. The requirements may also be non-functional such as System Security, Response time etc.**

2.2.4.2. Plan the specifications: Working with stakeholders to identify criteria after you have defined the project. The UML(Unified Modeling Language) diagrams can be used to present the work of new features and show how to apply it to your existing system. All UML diagrams will be presented later in research.

2.2.4.3. Construction / iteration: Work begins when the team defines the requirements. Designers and developers begin to work on their project, which is aimed at delivering a working product. The product must undergo different stages of development and it includes simple, minimal features.

2.2.4.4. Testing: The quality assurance team is reviewing the performance of the Pharmacy Management System in this process, and is searching for the error.

2.2.4.5. Deployment: In this process, the team is providing a system for the work environment of the customer.

2.2.4.6. Feedback: The last step after the project is released is feedback. The team is input on the product in this, and operates with input.

### 2.2.5 V-Shaped Model

The V-model is a type of SDLC model where process executes in V-shape sequentially. Also known as the Model of Verification and Validation. It is based on the association of a test step for every corresponding stage of growth. Design of each stage linked directly to the test process. The next step begins only after the completion of the preceding process, i.e. there is a corresponding testing activity for each creation operation.

***Verification:* This includes the technique of static analysis (review) that is performed without code execution. It is the product development phase assessment process to determine whether the specified requirements meet.**

*Validation*: involves the technique of dynamic (functional, non-functional) analysis, testing by executing code. Validation is the software validation process after completion of the development phase to assess if the program meets the specifications and requirements of the customers. So, V-Model contains Verification phases on one side on the other side of the Validation phases. Phases of verification and validation are joined by V-shape coding phase. Therefore, it is called a V-Model.

#### 2.2.5.1 Design Phase:

* + Requirement Analysis: This step involves extensive contact with the customer to understand their needs and specifications. This stage is known as the Gathering of Requirements, and similar to the previous models, **the requirements analysis of the Pharmacy Managements System will include functionalities such as adding and removing medicine from storage, calculating profit, revenue and cost. The requirements may also be non-functional such as System Security, Response time etc.**
  + System Design: In this phase the system design and the complete hardware and communication setup for developing product is carried out. **Also similar to the previous models,** implementation of the basic functionalities of the system will be implemented using JAVA Language later in this research.
* Architectural Design: The system design is further broken down into modules, which take on various functionalities. There is a clear understanding of how the data will be transferred and communicated between the internal modules and with the outside world.
* Module Design: In this phase, the system breaks down into small modules. It defines the specific configuration of the modules, also known as the Low-Level Design (LLD).

#### 2.2.5.2 Testing Phases:

• Unit Testing: Unit Testing Plans evolve during the design process of the module. Those Unit Test Plans are executed to remove code or unit level bugs.

• Integration testing: After the completion of the unit testing, the integration testing phase comes, in this phase, the modules are incorporated in integration testing, and the system is tested. Integration checking is carried out at the design phase of the Architecture. This test verifies the reciprocal contact of the modules.

• System testing: Testing the entire application with its functionality, interdependence and communication. It measures the functional and non-functional specifications of the application that was developed.

• User Acceptance Testing (UAT): UAT is conducted in a manufacturing environment close to that of the production. UAT verifies that the device provided meets the specifications of the customer, and that the device is ready for use in real world.

# 3.0 Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. Before development is initiated, these estimates are needed but how is this done? Several estimation procedures were developed and have common attributes. Cost estimation is useful during the planning stage, when one needs to choose how many engineers are required for the project and to develop a schedule.

Applying 3 different cost estimation models

## 3.1 COCOMO II Model

COCOMO II strategic four main elements are:

* Preserve original COCOMO diversity.
* Key the COCOMO II structure to the future marketplace software sectors described above.
* Key the COCOMO II sub model inputs and outputs to the level of information available.
* Make COCOMO II sub models specific to the required strategy of a project.

COCOMO II follows the principles of openness employed in the original COCOMO. So all its relationships and algorithms are going to be open to the public. In addition, all its interfaces are built to be available, well defined and parametric, so complementary preprocessors (models for analogy, case-based or other size estimation), project dynamics models, risk analyzers. Furthermore, packages of higher levels, for instance, project management packages, product management packages. In addition, COCOMO II can simply combine Negotiation Aids. To benefit the above-mentioned tech marketplace sectors COCOMO II provides a family of ever more detailed cost estimation models for software, tailored to the specific needs of each sector and type of information available to support cost estimates for the software.

The COCOMO II model for the Software Engineering sector is based on points. These points are basically a calculation of the number of the screens pages, reports and the high level programming language modules developed in the application, each having a weight by a three-level complexity factor. This is in line with the level of information generally known about an Application Composition product during its planning phases, and the corresponding level of accuracy required for its software cost estimates (such applications are generally developed in a few weeks to months by a small team). COCOMO II’s capacity to estimate Application Generator, System Integration, or Infrastructure developments is based on a tailor-made mix of the Application Composition Model (for initial prototyping efforts) and two accurate estimation models for successive life cycle segments, Early Design and Software Architecture.

* PERS = 1.2
* RCPX = 0.9
* RUSE = 0.95
* PDIF = 1.1
* PREX = 1
* SCED = 1.25
* FCIL = 1.15

M=1.2x0.9x0.95x1.1x1x1.25x1.15 = 1.62

Size = 6.5 KLOC

Rate=4000$/pm

Estimated cost =4000x45=180,000$

## 3.2 Line of Code Model

In a software development project, writing code typically amounts only to a small portion of spent time and effort. The other larger portion, however, is spent on analyzing the problem and trying different approaches to solve it. Planning and discussing ideas, learning, reading documentation, reverse engineering, trial and error, solving issues with libraries, testing, fixing bugs, setting up infrastructure, preparing different environments, automation, CI/CD pipelines, communication, meetings and reviewing codes are also time consuming. Additionally, the code itself is likely to be refactored, discarded and rewritten several times before it settles down. The code can be bad or good and 100 lines of good code can be much better than 1000 lines of bad code. The code can be simple or complex; a line of complex code can take 100 times as much time and effort as a line of simple code. Bottom line, LOC is an inaccurate metric. LOC is more useful as a quick and rough estimate of time needed to learn an existing project when a developer starts to work on it without previous experience.

LOC= 6500

Productivity=150 loc/pm

Rate=4000$/pm

Effort = 6500/150 = 44 pm

Estimated cost = 44x4000=176,000$

## 3.3 Functional Points Model

From a practical perspective, function points come into play in Software Project Management and you are most likely to encounter this when it comes to Software Cost Estimation. Function points are alternatives to LOC (lines of code), but that is in a very broad sense. Both LOC and FP are used as software metrics. Both are used to approximate software cost. However, they are very different in what they measure/indicate. LOC is a very simplistic view counting lines of code. Function points are used to represent “the amount of business functionality the product delivers to the user”. Clearly, the higher is the functionality you provide the client, the higher would you want to charge. There is a very dominant business perspective in the measurement of function points and it became widely popular due to its focus on the customer. Somewhere the method works around “You pay for what you want and get, not what I ended up developing” since it measures the functionality the user has requested for and received. Besides helping with cost estimation, FP also help customers evaluate the benefit of a software product by counting functions that match their requirements. Thus, the efficiency of this method is heavily dependent on correct requirements analysis and documentation.

METHOD TO COMPUTE FUNCTION POINTS:

Several standards and frameworks allow for measuring function points. Over the years, many changes and additions have been made to Function Point Analysis and today it is a large field with specialized third party organizations offering services and project management software that allow estimation by various standards. However, broadly speaking, the following method captures the essence:

1. The counting boundary is established as a border to separate the current project being measured from external applications.

This helps identify which functions will be included in the count.

2. Next, to determine Unadjusted Function Point Count (UFP), you need

i) Count of data functions

They are of two types:

* ILF (Internal Logical Files): data or control information residing within the application, user is responsible for its maintenance.
* EIF (External Interface File): data or control information used only for exchange residing on other systems, user is not responsible for maintaining this information.

ii) Count of transactional functions

There are the following three types:

* EI (External Input) : methods for maintaining ILF (add, update, delete etc.) Help introduce changes in system’s internal data within boundary
* EO (External Output): methods for producing output, i.e. internal data can be presented outside system boundary
* EQ (External Queries): methods to read system data without modifying it.

These five types of function data are then ranked as High, Average or Low based on standards that prescribe the framework for doing so. Since there are various methods proposed by different organizations, one could say that FP measurement can vary subjectively.

3. Calculate value adjustment factor by evaluating various system characteristics. Here again, methods and standard vary. A sample can be:

4. The function point is obtained as

FP= UFP\*VAF

FP= FUNCTION POINT

UAF= UNADJUSTED FUNCTION POINT

VAF= VALUE ADJUSTMENT FACTOR.

|  |  |  |  |
| --- | --- | --- | --- |
| **Measurement parameter** | **Count** | **Weighing factor** | **Total** |
| Number of external inputs (EI) | 50 | 3 | 150 |
| Number of external outputs (EO) | 40 | 5 | 200 |
| Number of external inquiries (EQ) | 12 | 3 | 36 |
| Number of internal files (ILF) | 7 | 15 | 105 |
| Number of external interfaces (EIF) | 5 | 7 | 35 |
| Total Count | 114 | Total | 526 |

|  |  |
| --- | --- |
| Backup and recovery | 4 |
| Data communication | 5 |
| Distributed processing functions | 2 |
| Is performance critical? | 1 |
| Existing operating environment | 4 |
| On-line data entry | 0 |
| Input transaction built over multiple screens | 3 |
| Master files updated on-line | 2 |
| Complexity of inputs, outputs, files, inquiries | 4 |
| Complexity of processing | 3 |
| Code design for re-use | 1 |
| Are conversion/installation included in design? | 0 |
| Multiple installations | 5 |
| Application designed to facilitate change by the user | 2 |
| Total | 38 |

Productivity= 12 fp/pm

Effort=531/12 = 44.25 pm

Rate = 4000 $/pm

Estimated cost =4000 x 44.25=177,000$

# 4.0 Software Architecture

Software architecture refers to the basic constructs of a software system and the practice of designing frameworks and systems like these. Every structure comprises elements of the program, relationships between them, and properties of both elements and relationships. Software architecture strives to reduce the risk of costly changes to the software after implementation. Choices for software architecture include specific structural options from possible options in the software design.

## 4.1 Object-Oriented Architecture

4.1.1 What is Object-Oriented Architecture Object Oriented Architecture is a significant term for the software development. It is a design model focused on separating responsibilities for an application or device into individual objects, which can be reused and self-sufficient. The common object-oriented design approach is seeing a software system as a set of entities known as objects. Object oriented is based on the representation of events in the real world.

**Object-Oriented Architecture Advantages**

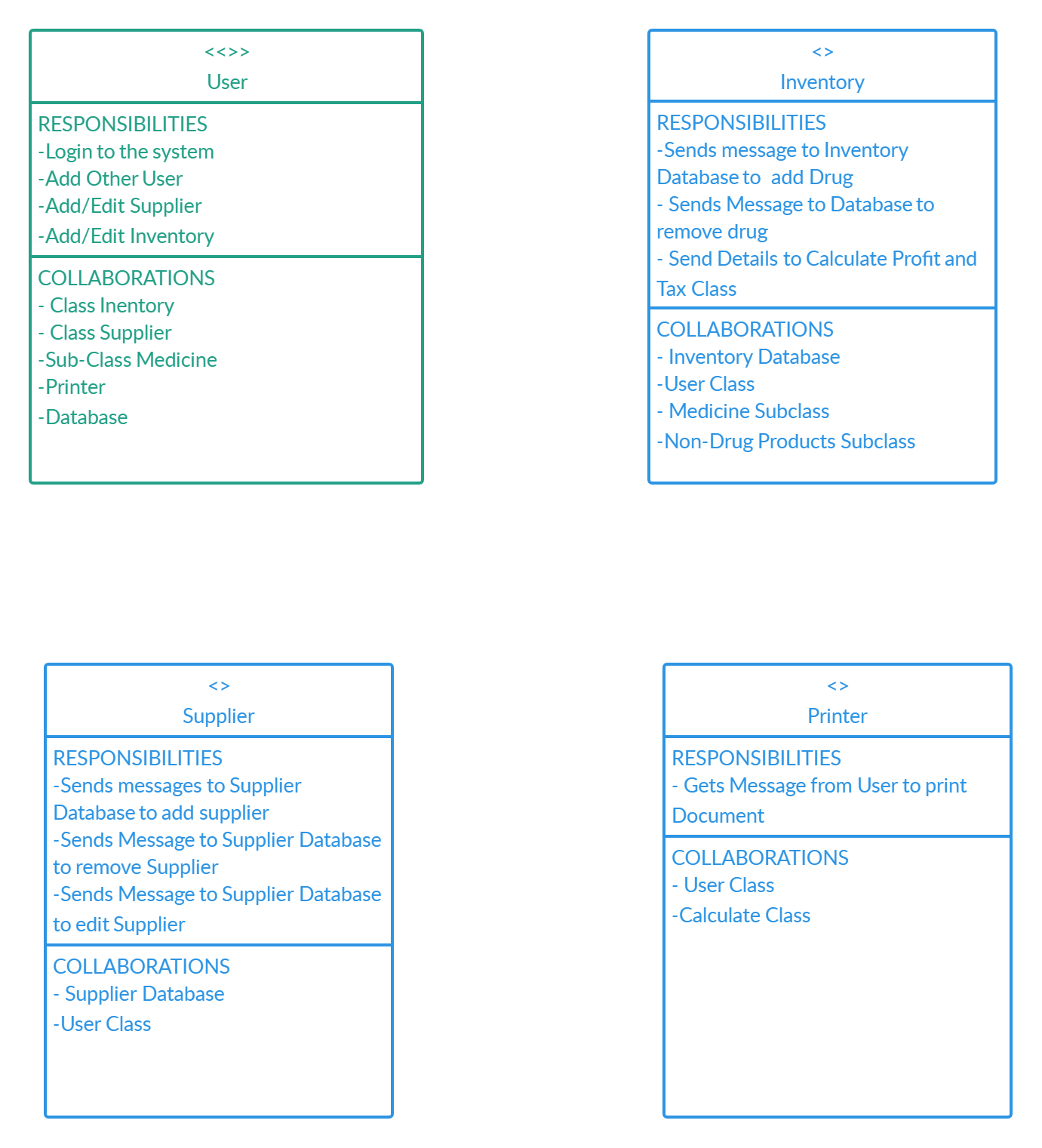
* Object-Oriented Architecture maps the program to objects in the real world to make it more comprehensible.
* Because of software reuse, it is easy to manage and increases device efficiency.
* With polymorphism and abstraction, this architecture offers reusability.
* During execution, it has the ability to handle the errors. (Stronghold)
* It has the potential to add new functionality, which does not affect the system.
* Encapsulation increases testability.
* Object-Oriented architecture reduces the time and cost of growth.

**Impairment in Object-Oriented Architecture**

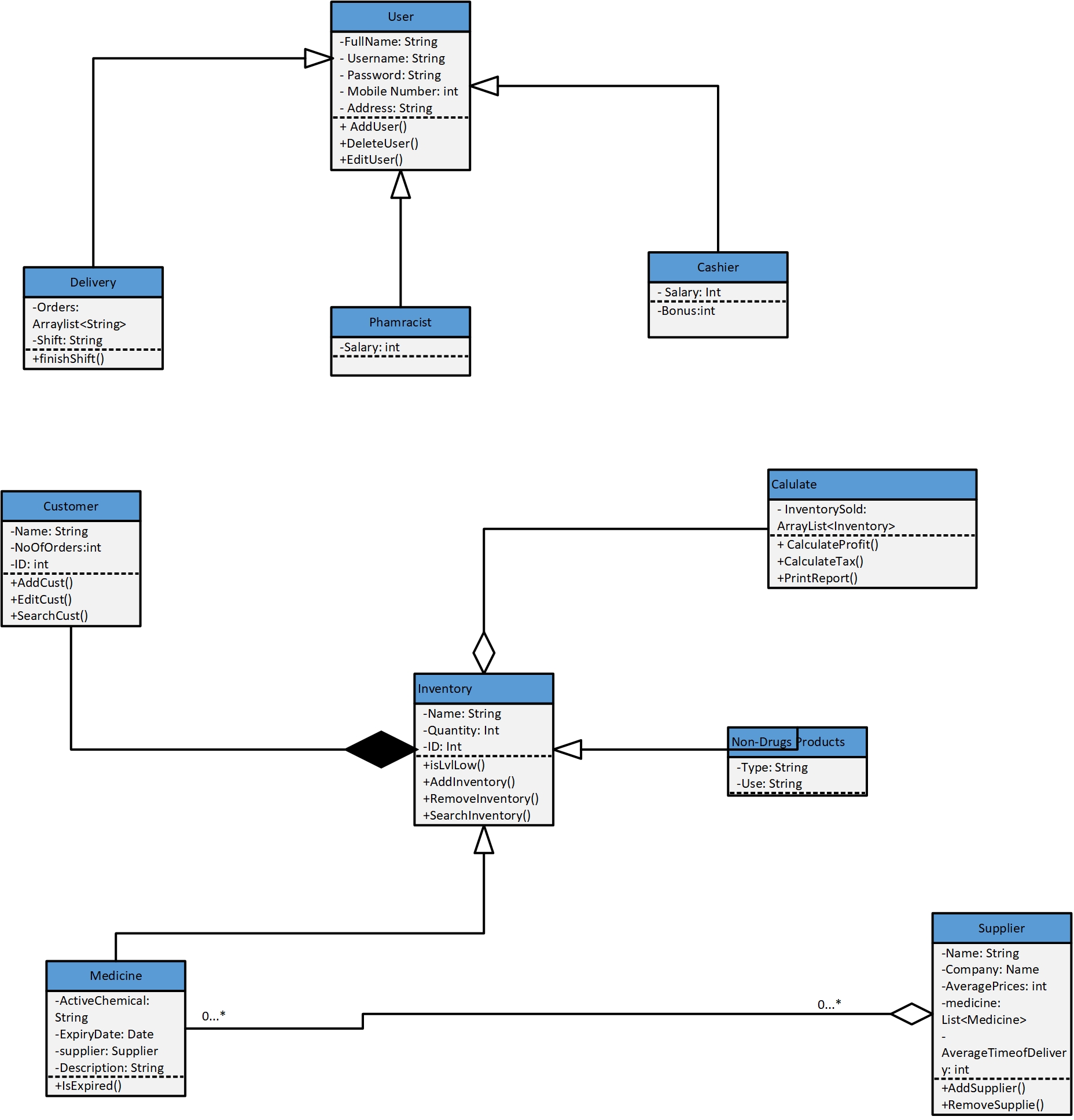
* Object-oriented architecture has difficulties in deciding all of the classes and artifacts required for a program.
* Since it is object based, it is difficult to complete a solution within projected time and budget
* Without explicit reuse procedure, this methodology does not lead to successful reuse on a large scale.

### 4.1.2 Noun Extraction and CRC Cards

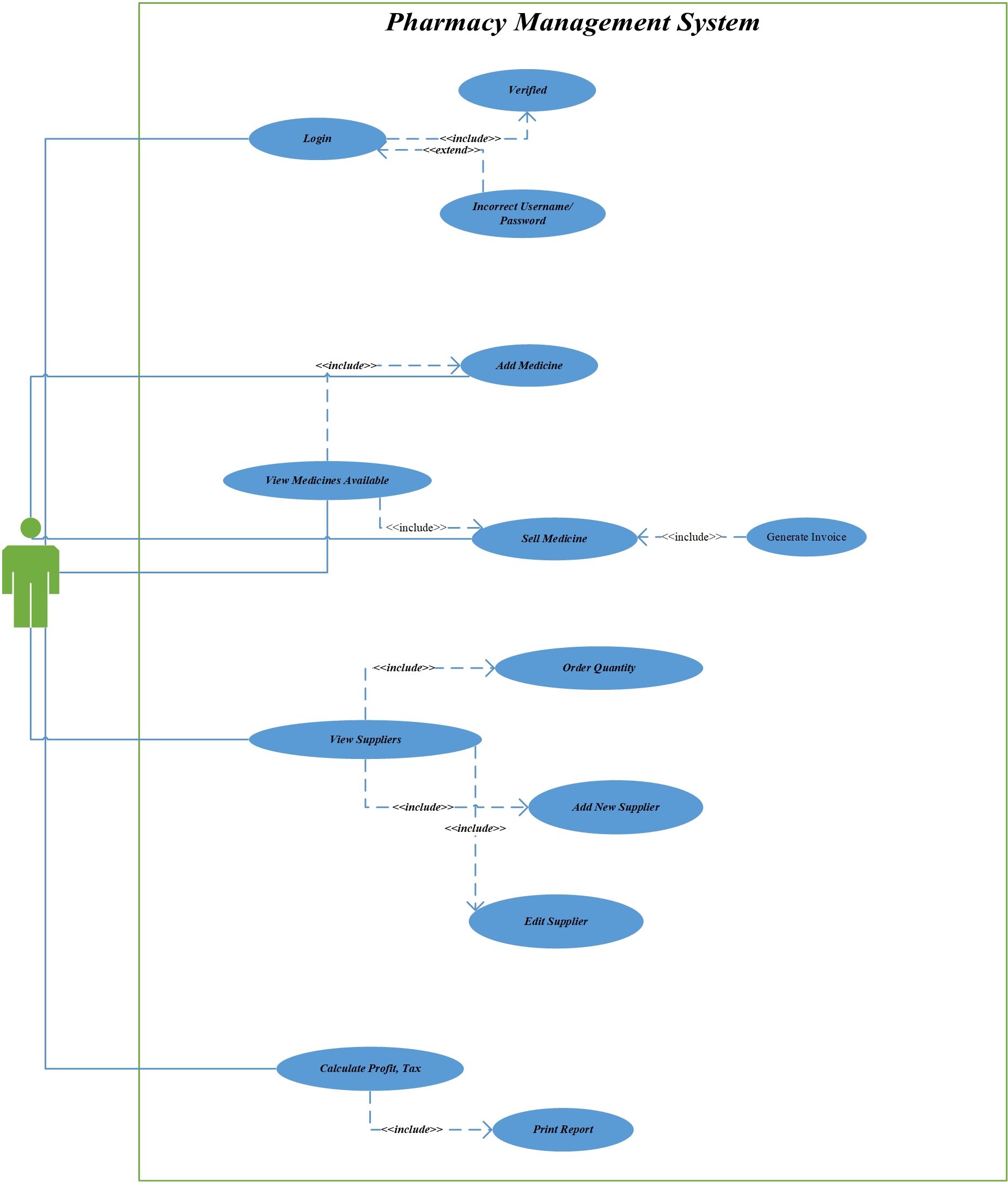
Noun extraction and CRC Cards ae very useful when it comes to Software architecture as they pave the way to creating class diagram which is considered the main pillar on which object oriented programming is based on. Noun extraction provides the system’s main nouns such as Pharmacy, Inventory, User, Delivery, Supplier etc. CRC Cards take these nouns and convert them into a detailed, yet simple diagram that creates a visually simple way to understand the main classes in the system. CRC (Class Responsibility Collaboration) provides the class’s responsibilities as in what the class should do in the system. In addition, it also provides the classes that this class interacts and collaborates with. The following is the CRC Diagram for the Pharmacy Management System.



### 4.1.3 Pharmacy Management System UML Class Diagram



### 4.1.4 Use Case Diagram



### 4.1.5 Use Case Description

**Use Case Name Login**

Related Requirements Requirement A1

Goal in Context Users details need to be validated

Preconditions Appropriate proof of identity

Successful End Conditions The details are verified

Failed End Conditions The details are not verified

Primary Actor User

Secondary Actor None

Trigger User Provides Credentials

Main Flow 1. The user provides credentials

2. Credentials Database verifies data

3. Details returned as verified

Extensions 1. User provides wrong details

2. Details returned as unverified

3. A message is returned to try again

**Use Case Name Add Medicine**

Related Requirements Requirement A2

Goal in Context Add medicine to Database

Preconditions Medicine Information entered

Successful End Conditions Medicine added to Database

Failed End Conditions Database fails to record medicine’s information

Primary Actor User

Secondary Actor none

Trigger Pharmacist provides medicine’s information

Main Flow 1. User chooses “Add Medicine” from Menu

2. User enters medicine’s details

3. The new medicine is stored in the Inventory

Database

Extensions 1. User enters information in wrong fields, eg: User

enters letters in expiration date field.

2. Database fails to record the new medicine.

3. A message is returned for wrong type of

information entered

**Use Case Name Order Known Quantity**

Related Requirements Requirement A3

Goal in Context Contacting Supplier with the required quantity

Preconditions Quantity of medicine is below a critical level

Successful End Conditions An email is sent to the supplier

Failed End Conditions The application does not send an email to the

supplier

Primary Actor User

Secondary Actor none

Trigger Stock level of a medicine is critically low

Main Flow 1. Database signals that a level of stock is low

2. User is notified

3. User chooses the quantity to order

4. An Email is sent to the supplier with the quantity

Extensions 1. Database fails to signal that a level of stock is low

2. User is not notified

3. Medicine runs out of stock

**Use Case Name Calculate Profit, Tax**

Related Requirements Requirement A4

Goal in Context Calculating the Pharmacy’s Profits and Taxes

Preconditions Full documentation about inventory prices and quantities

Successful End Conditions Correct Calculations are made and documented

Failed End Conditions Wrong or no calculations are made

Primary Actor User

Secondary Actor none

Trigger User choses “Calculate Profit and Taxes”

Main Flow 1. Every medicine sold or bought is recorded on the software

2. User choses Calculate Profit and Taxes from menu

3. A full Tax and Profit report is made

Extensions 1. Not every medicine sold or bought is recorded

2. User Choses Calculate Profit and Tax

3. A false and inaccurate report is made

**Use Case Name Sell Medicine**

Related Requirements Requirement A2

Goal in Context Remove Medicine from Database

Preconditions Medicine is sold or expired

Successful End Conditions Medicine is removed from database

Failed End Conditions Medicine is still recorded in the Database

Primary Actor User

Secondary Actor none

Trigger Cashier sensor detects a barcode

Main Flow 1. Cashier sensor detects a barcode

2. Medicine in Inventory database is reduced by 1

3. Medicine is recorded as sold for report calculations

Extensions 1. Cashier sensor fails to detect a barcode

2. Cashier enters inventory details manually.

3. Database fails to reduce medicine’s quantity

**Use Case Name Add New Supplier**

Related Requirements Requirement A4

Goal in Context Add a new supplier to the database

Preconditions User enters supplier’s details

Successful End Conditions Database record suppliers details

Failed End Conditions Database does not record supplier’s details

Primary Actor User

Secondary Actor none

Trigger User provides supplier’s full details

Main Flow 1. User choses to add new supplier

2. User enters supplier’s information

3. Database saves the information

Extensions 1. User choses to add new supplier

2. User enters supplier’s information

3. Database does not save the information

**Use Case Name Print Report**

Related Requirements Requirement A5

Goal in Context Printing a full report on profit and taxes

Preconditions Inventory sold details are provided

Successful End Conditions A full report is made and printed

Failed End Conditions A false or inaccurate report is made

Primary Actor User

Secondary Actor none

Trigger Profit and Taxes are calculated

Main Flow 1. Program to calculate profit and taxes runs successfully

2. Printer in online and there is enough paper

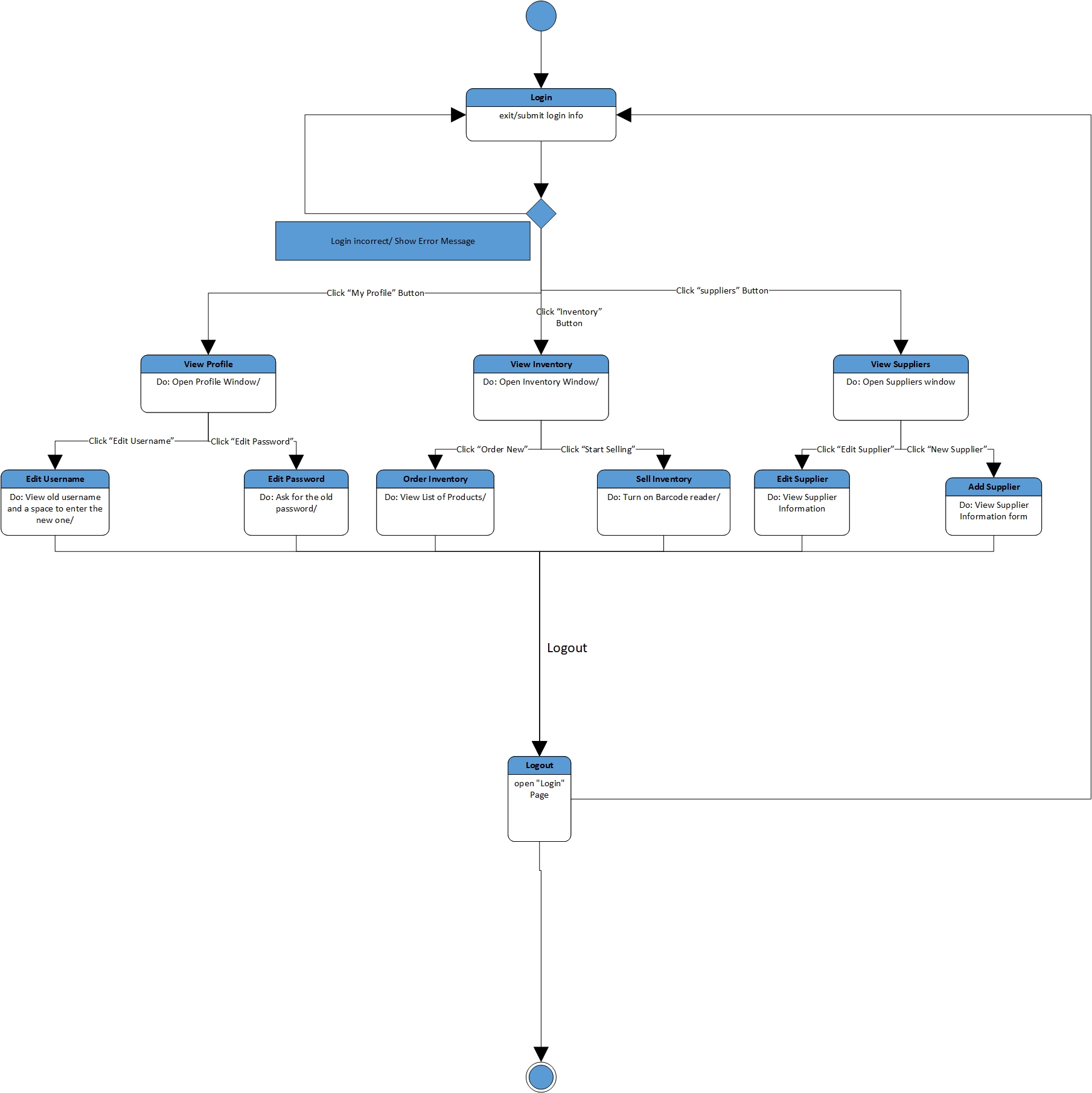
3. Program sends the printer the document to print

Extensions 1. Program to calculate profit and taxes runs successfully

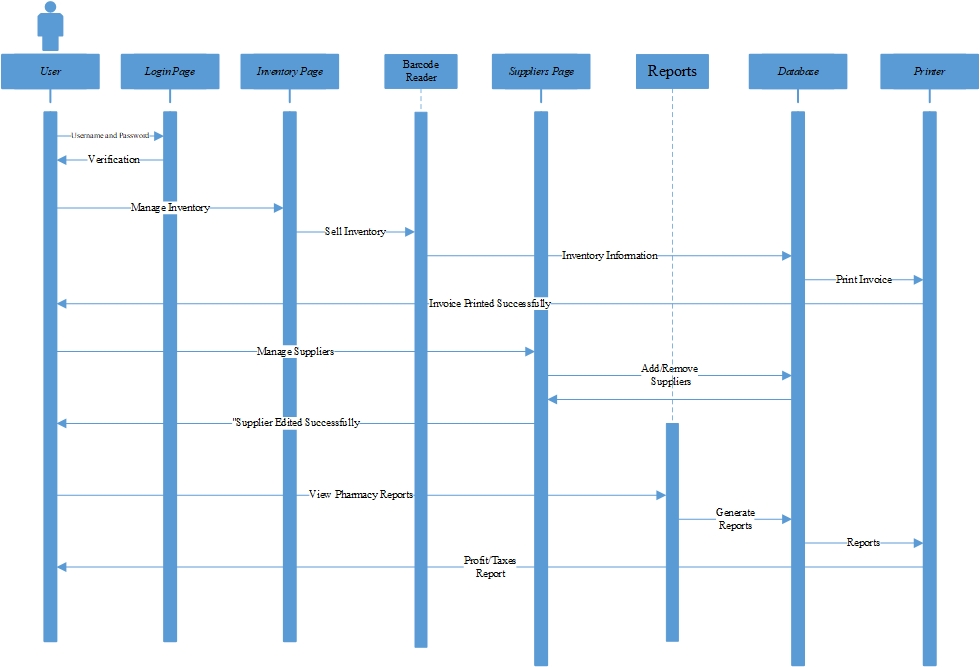
2. Paper is jammed or not enough ink is present

3. User is notified to add ink or unjam the paper

### 4.1.6 UML – State Diagram



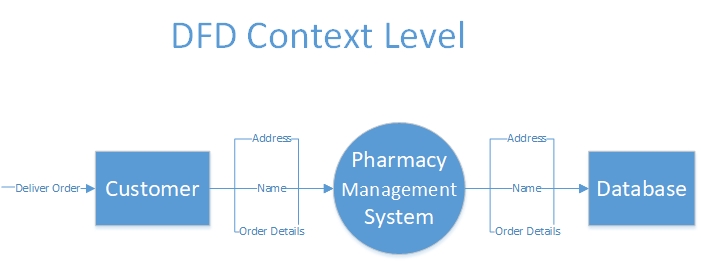
### 4.1.7 UML – Sequence Diagram



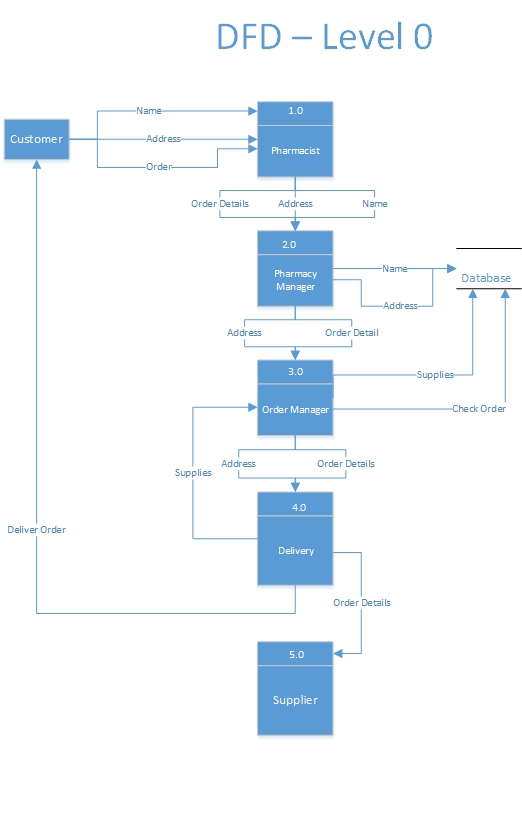
## 4.2 Layered Architecture

Multi-layered architecture of software is among today's most common architectural trends. The increasing complexity of modern applications is moderated by this. It also encourages a more agile way of working. That is important when you consider today's DevOps dominance and other similar methodologies. Often called tiered architecture, or n-tier architecture, a multi-layered architecture of software consists of various tiers, each corresponding to a different function or integration. Making changes is easier as each layer is separate and independent.

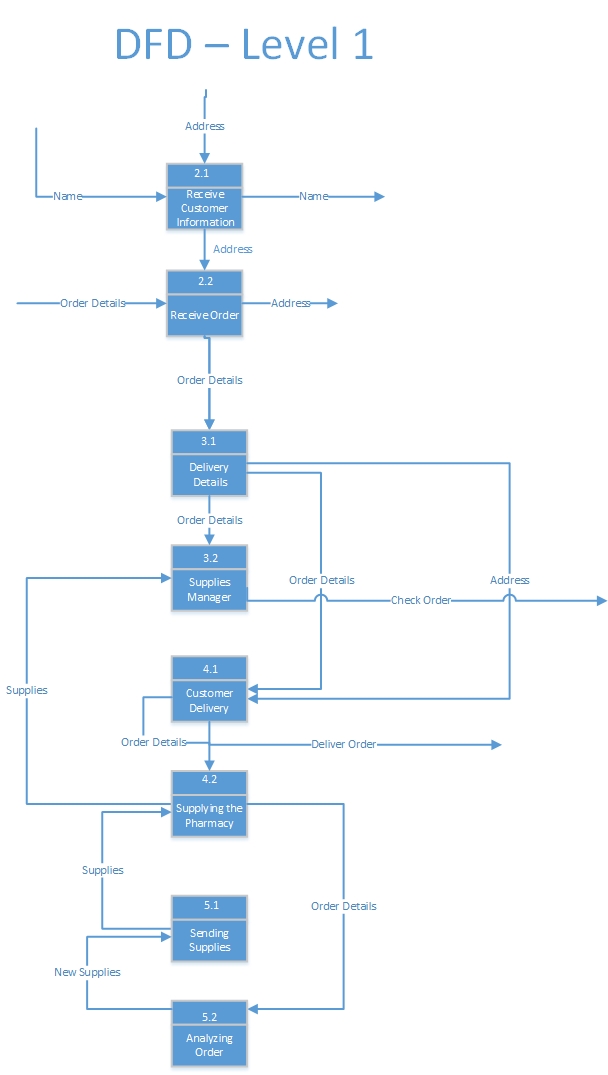
### 4.2.1 Data Flow Diagram - DFD Context



### 4.2.2 DFD – Level 0

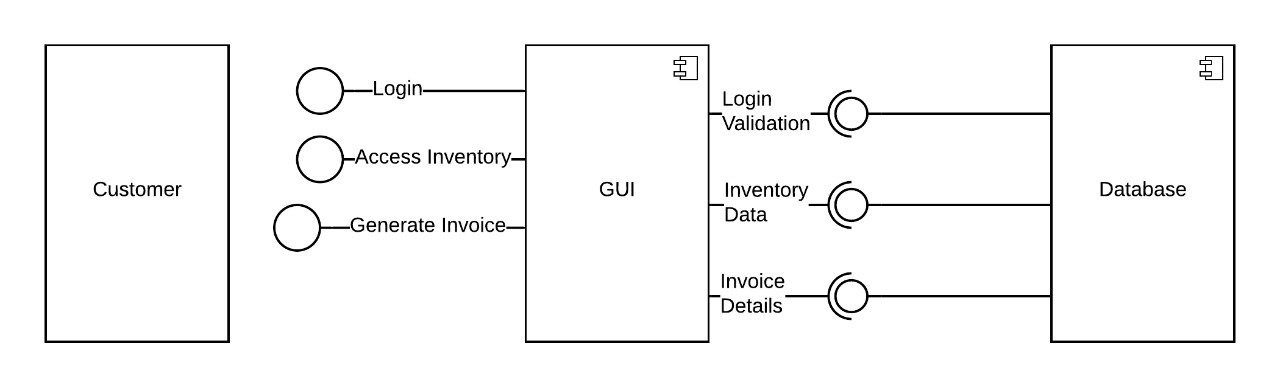


### 4.2.3 DFD – Level 1

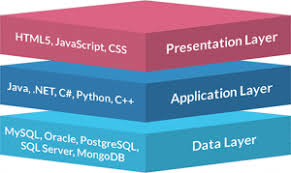


### 4.2.4 Component Diagram

Components are mainly used to create reusable pieces of code. Components are combined to create whole programs and they perform operations interact with classes and implement interfaces; however, components tend to perform more complicated tasks than you normally would with a regular class.



## **4.3 Merged Styles Architecture**

 To get the best of both architecture styles, a merged architecture style will be used, therefore using the concepts of layered architecture to divide the system into layers and use classes to implement the layers in an object-oriented way. The diagram below will illustrate the idea of the layers used in the system.

The Data Layer contains the database that is responsible for maintaining and storing the data of the inventory, users, and suppliers. The Application Layer contains the functions that call the database to retrieve or store information. Not only that, but the application layer communicates with the presentation layer to know which functions to be called as the user executes an event. The Presentation Layer is the layer visible to the user and therefore must be styled and easy-to-use. Inside these layers, Classes and Use Cases will be used to create them. The Application layer will contain the main classes of the system including their functions. The Presentation Layer will contain the GUI Classes used to provide the graphical look to the user.

This merged architecture style will benefit the system in many ways as it benefits from the Object-Oriented Architecture being loosely coupled so that classes may be implemented without affecting other classes and the reusability of these classes. Furthermore, the system will benefit from the Layered Architecture, as it will support incremental development of layers.

## 4.4 Cohesion and Coupling

### 4.4.1 Coupling.

An example of the frequency of Program Unit interconnections. If the modules are highly coupled, they have program units that depend on each other. Loosely coupled, they are composed of independent or almost independent units. Modules are independent if they can fully function without the other being present. Obviously, modules cannot be completely independent of each other. Must interact, so that desired output can be produced.

In order to minimize the number of interfaces between the modules, minimize each interface's complexity and control the type of info flow, a module interface is used to pass information from and to other modules. Generally speaking, modules are closely coupled when using shared variables or sharing control data. Loose coupling when information is kept within a unit and interact with other units through parameter lists. Tight coupling, if global data is exchanged.

Do not pass the entire record if you need just one area of a record. Keep GUI as simple as possible and as small as possible.

Info flow of two types: data or control.

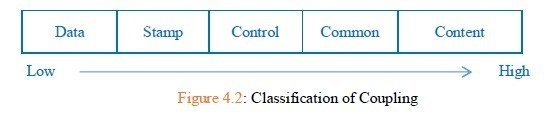
Data coupling: Parameter lists are used to transfer data objects between routes

Passing or receiving back control information means the module's action will depend on this control information, which makes it difficult to understand the module.

Data-only interfaces result in the lowest degree of coupling, followed by interfaces that only transfer control data.

Main types of coupling.

* Data Coupling: If the modules rely on the fact that they interact by passing data only, then the modules are said to be coupled with data. The components are independent of each other in data coupling and are communicated through data.
* Stamp Coupling: Two modules are “Stamply” coupled if they communicate via a passed data structure, which contains more information than the required.
* Control Coupling: If the modules communicate by passing control information, then it is said that they are control coupled. It can be bad if parameters mean entirely different behavior and good if parameters allow functionality to be reused. Example: A function that takes another function as parameters.
* External Coupling: The modules rely on other modules in external coupling, whether external to the software being built or to a particular form of hardware.
* Common Coupling: The modules share data such as global data structures. Changes in global data mean tracing back to all modules that access the data to determine the change's impact. Consequently, it has disadvantages such as difficulty in reusing modules, reduced ability to control access to data and reduced maintenance.
* Content Coupling: It is considered the worst type of coupling as the module can influence the date of another module.



### 4.4.2 Cohesion

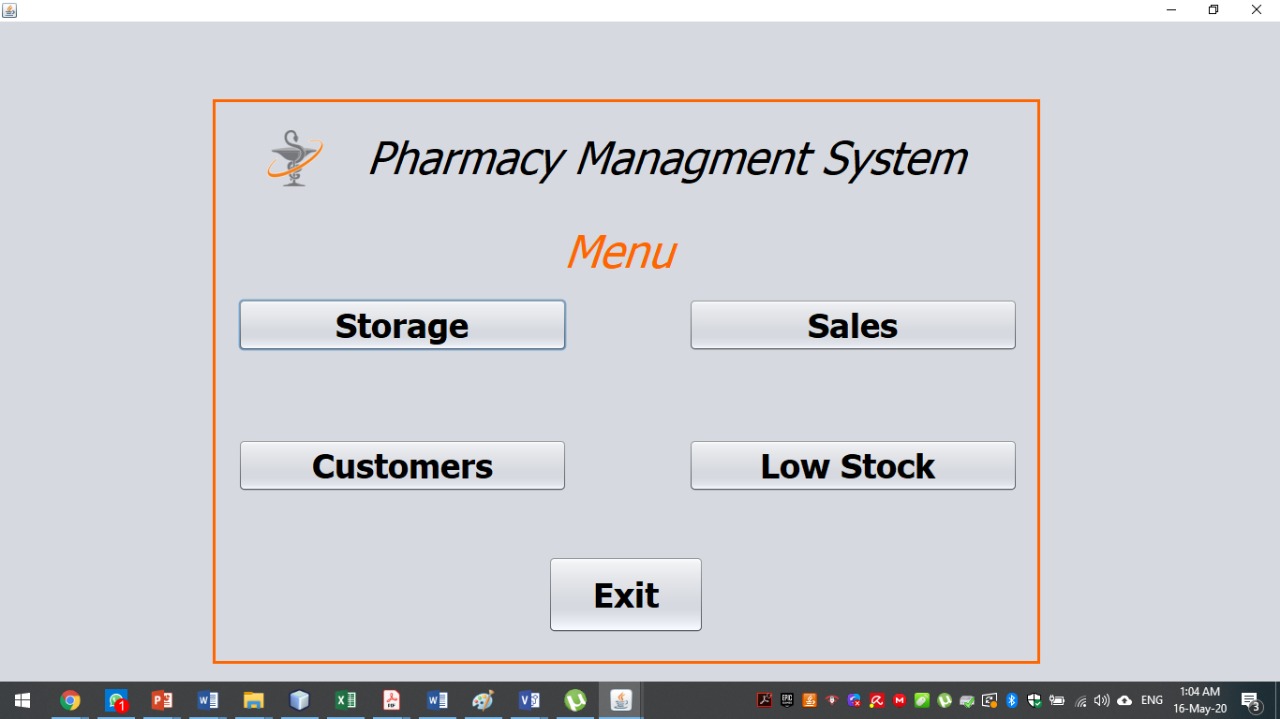
* Cohesion is an indicator of module relationship.
* Cohesion shows the relative strength of function of the module.
* Cohesion is a degree (quality) to which the single thing is focused by a component / module.
* In designing, you should aim for high consistency i.e. a coherent component / module focused on a single task with little contact with other device modules.
* Cohesion is the kind of natural extension of data hiding, class making all members visible with the default visibility of a packet.

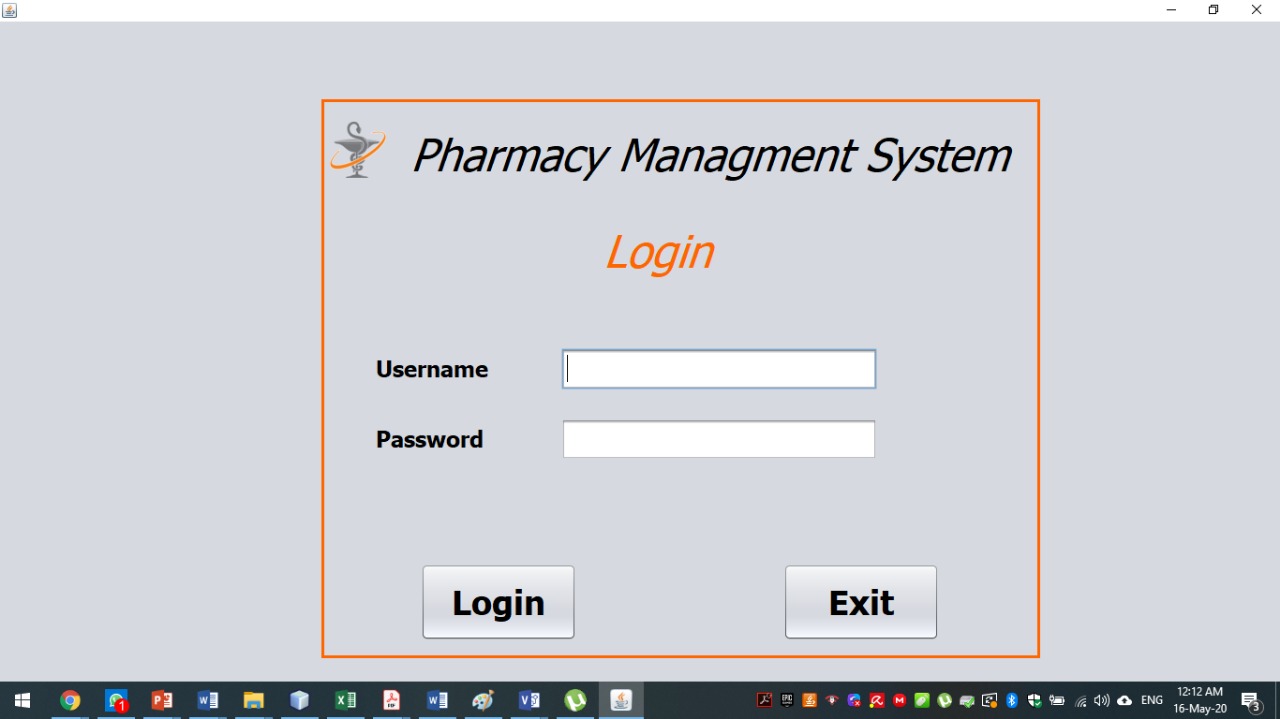


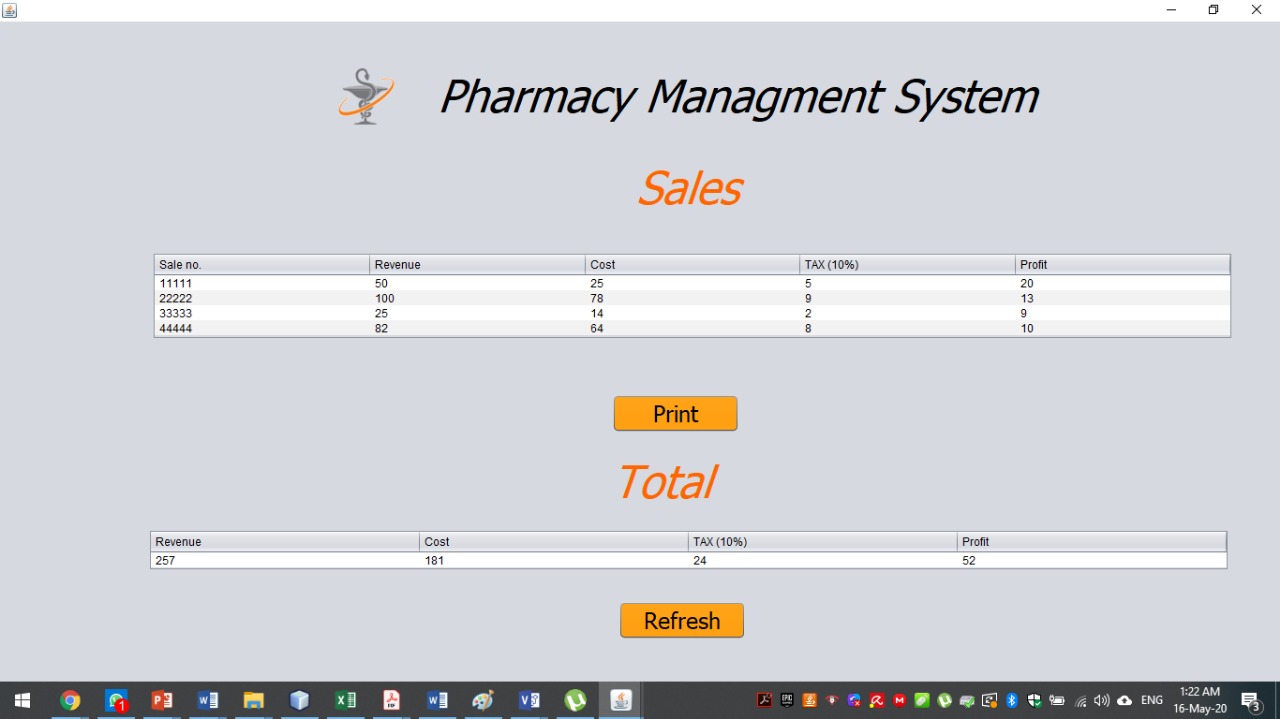
Types of cohesion

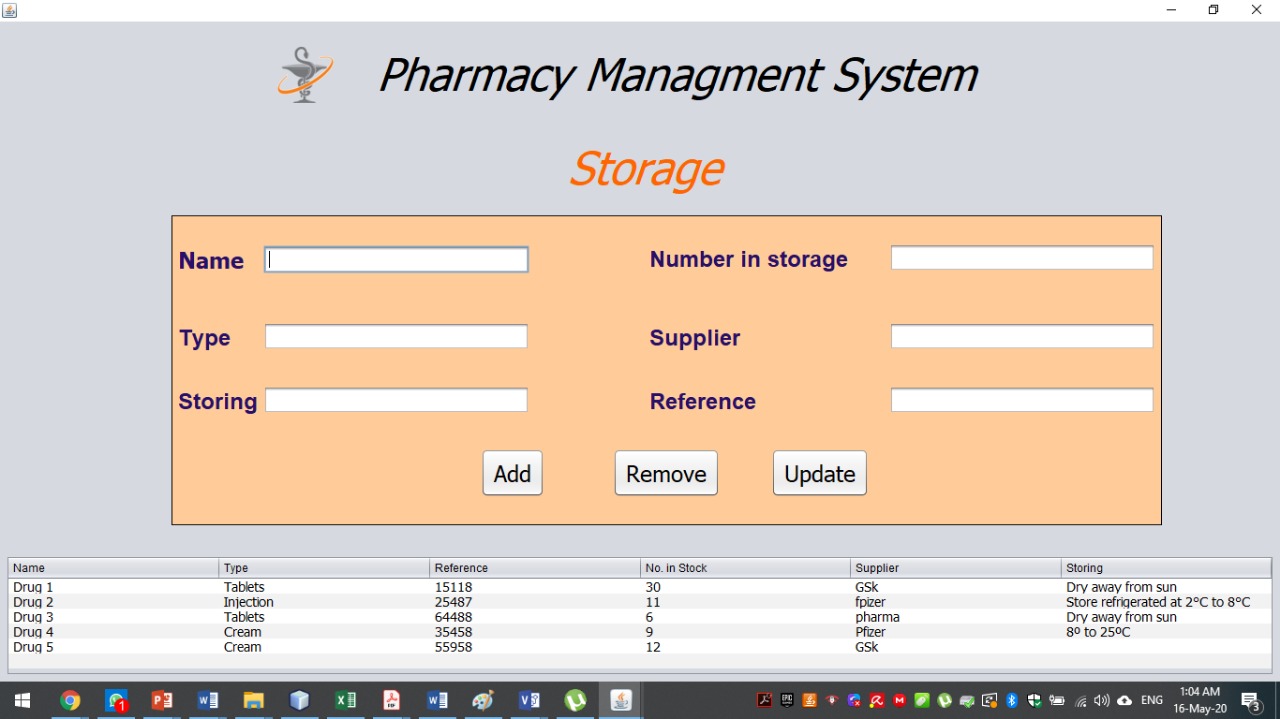
* Functional Cohesion: The part contains every critical element for a single computation. The mission and the roles perform a logical unity. It is an ideal location.
* Sequential Cohesion: Some data is generated by an element, which becomes the input for another element, i.e. data flow between sections. It occurs in functional programming languages, naturally.
* Communicational cohesion: Two components act on the same data input or contribute to the same data output. For example, update the database record and send it to the printer.
* Procedural consistency: Procedural consistency components compensate for order of execution. Actions also have poor links and are unlikely to be reusable. Ex- Executing functions that calculate, print, record the GPA of a student.
* Temporal cohesion: This type of cohesion involves functions that are executed during the same time-span. Different activities occur all at the same time.
* Logical cohesion: The components are logically and not functionally related. Ex- A portion reads Tape, Disk, and Network inputs. For these functions, all code is in the same section. Operations are connected but the functions vary considerably.
* Coincidental Cohesion: The components are unrelated to each other. The elements have no conceptual relationship in source code other than the location. It is the worst form of cohesion, as the modules are not related. Ex-print next line and reverse a string’s characters into a single part.

# 5.0 GUI









# 6.0 References

* Neil Maiden, Sarah Jones. "Agile Requirements: Can We Have Our Cake and Eat It Too?", IEEE Software May/June 2010, pp.87-88.
* Bernd Oestereich. Developing Software with UML: Object-Oriented Analysis and Design in Practice. Addison-Wesley Pub Co; 1st edition (March 1, 1999).
* David Janzen, Hossein Saiedian. "Test-Driven Development: Concepts, Taxonomy, and Future Directions", IEEE Software, 38(9):43-50 (September 2005).
* Mark W. Maler, David Emery, and Rich Hilliard. "Software Architecture: Introducing IEEE Standard 1471", IEEE Computer, April 2001.
* Watss S. Humphrey. "Why Can't We Manage Large Projects?", CrossTalk: The Journal of Defense Software Engineering, Jul/Aug 2010.
* Miyazaki, Y., and K. Mori (1985), "COCOMO Evaluation and Tailoring," Proceedings, ICSE 8, IEEE-ACM-BCS, London, August 1985, pp. 292-299.