PATIENT MANAGEMENT SYSTEM

FOR MEDICAL DISPENSARIES

OBJECT ORIENTED SOFTWARE ENGINEERING MINI PROJECT – SEMESTER VI

CERTIFICATE

This is to certify that the following students of Third Year (Semester VI) Computer Engineering 2012-13:

- Sinai Nadkarni Viren – 101105064
- Ricky O-Shangpliang – 101105055
- Shilpa Rana - 101105061

Undertook a project in the subject Object Oriented Software Engineering to develop a *Patient Management System for Medical Dispensaries* and has completed it satisfactorily and up to the desired standards.

Lecture-In-Charge	Head of Department	Principal

Date:

Place: Farmagudi, Goa

TABLE OF CONTENTS

INTRODUCTION

A software development process, also known as a software development life-cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a life-cycle model a more general term and a software development process a more specific term.

PLANNING

Planning is an objective of each and every activity, where we want to discover things that belong to the project. An important task in creating a software program is extracting the requirements or requirements analysis.[1] Customers typically have an abstract idea of what they want as an end result, but not what software should do. Skilled and experienced software engineers recognize incomplete, ambiguous, or even contradictory requirements at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

IMPLEMENTATION, TESTING AND DOCUMENTING

Implementation is the part of the process where software engineers actually program the code for the project.

Software testing is an integral and important phase of the software development process. This part of the process ensures that defects are recognized as soon as possible.

Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an API, be it external or internal. The software engineering process chosen by the developing team will determine how much internal documentation (if any) is necessary. Plan-driven models (e.g., Waterfall) generally produce more documentation than Agile models.

DEPLOYMENT AND MAINTENANCE

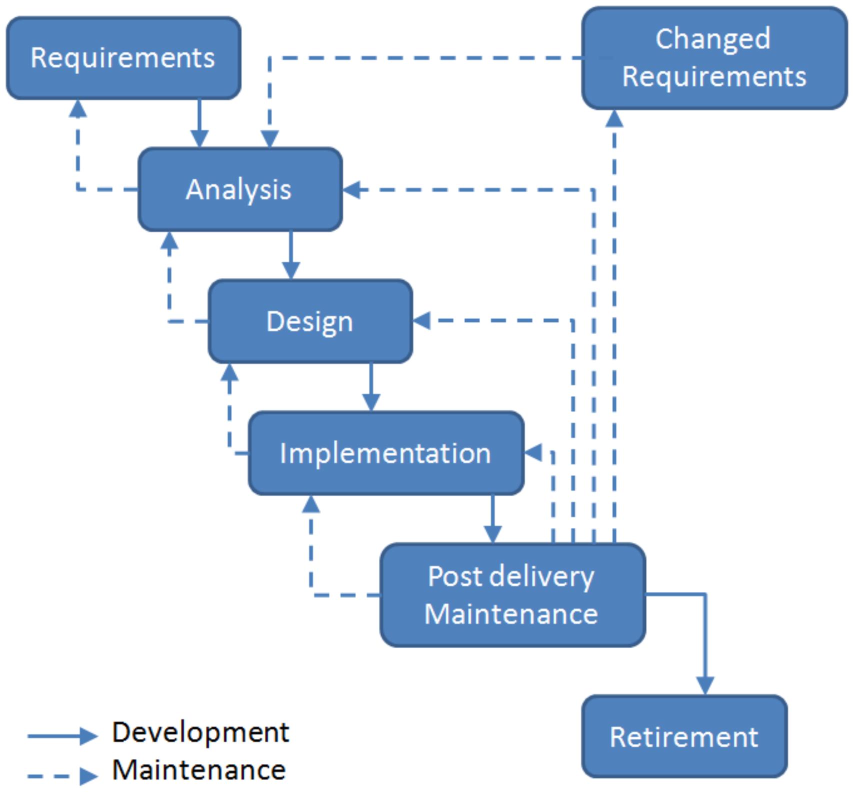
Deployment starts after the code is appropriately tested, approved for release, and sold or otherwise distributed into a production environment. This may involve installation, customization (such as by setting parameters to the customer's values), testing, and possibly an extended period of evaluation.

Software training and support is important, as software is only effective if it is used correctly.

Maintaining and enhancing software to cope with newly discovered faults or requirements can take substantial time and effort, as missed requirements may force redesign of the software.

THE WATERFALL MODEL

The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance.



The waterfall development model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

The first known presentation describing use of similar phases in software engineering was held by Herbert D. Benington at Symposium on advanced programming methods for digital computers on 29 June 1956. This presentation was about the development of software for SAGE. In 1983 the paper was republished with a foreword by Benington pointing out that the process was not in fact performed in a strict top-down fashion, but depended on a prototype.

The first formal description of the waterfall model is often cited as a 1970 article by Winston W. Royce, although Royce did not use the term "waterfall" in this article. Royce presented this model as an example of a flawed, non-working model. This, in fact, is how the term is generally used in writing about software development—to describe a critical view of a commonly used software development practice.

The waterfall model has the following phases that are followed in order:

1. Requirements specification
2. Design
3. Implementation
4. Integration
5. Testing and debugging
6. Installation
7. Maintenance

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. Various modified waterfall models (including Royce's final model), however, can include slight or major variations on this process.

In a strict Waterfall model, after each phase is finished, it proceeds to the next one. Reviews may occur before moving to the next phase which allows for the possibility of changes (which may involve a formal change control process). Reviews may also be employed to ensure that the phase is indeed complete; the phase completion criteria are often referred to as a "gate" that the project must pass through to move to the next phase. Waterfall discourages revisiting and revising any prior phase once it's complete. This "inflexibility" in a pure Waterfall model has been a source of criticism by supporters of other more "flexible" models.

Requirements analysis: This first step is also the most important, because it involves gathering information about what the customer needs and defining, in the clearest possible terms, the problem that the product is expected to solve. Analysis includes understanding the customer's business context and constraints, the functions the product must perform, the performance levels it must adhere to, and the external systems it must be compatible with. Techniques used to obtain this understanding include customer interviews, use cases, and "shopping lists" of software features. The results of the analysis are typically captured in a formal requirements specification, which serves as input to the next step.

Design: This step consists of "defining the hardware and software architecture, components, modules, interfaces, and data...to satisfy specified requirements" (Wikipedia). It involves defining the hardware and software architecture, specifying performance and security parameters, designing data storage containers and constraints, choosing the IDE and programming language, and indicating strategies to deal with issues such as exception handling, resource management and interface connectivity. This is also the stage at which user interface design is addressed, including issues relating to navigation and accessibility. The output of this stage is one or more design specifications, which are used in the next stage of implementation.

Implementation: This step consists of actually constructing the product as per the design specification(s) developed in the previous step. Typically, this step is performed by a development team consisting of programmers, interface designers and other specialists, using tools such as compilers, debuggers, interpreters and media editors. The output of this step is one or more product components, built according to a pre-defined coding standard and debugged, tested and integrated to satisfy the system architecture requirements. For projects involving a large team, version control is recommended to track changes to the code tree and revert to previous snapshots in case of problems.

Testing: In this stage, both individual components and the integrated whole are methodically verified to ensure that they are error-free and fully meet the requirements outlined in the first step. An independent quality assurance team defines "test cases" to evaluate whether the product fully or partially satisfies the requirements outlined in the first step. Three types of testing typically take place: unit testing of individual code modules; system testing of the integrated product; and acceptance testing, formally conducted by or on behalf of the customer. Defects, if found, are logged and feedback provided to the implementation team to enable correction. This is also the stage at which product documentation, such as a user manual, is prepared, reviewed and published.

Installation: This step occurs once the product has been tested and certified as fit for use, and involves preparing the system or product for installation and use at the customer site. Delivery may take place via the Internet or physical media, and the deliverable is typically tagged with a formal revision number to facilitate updates at a later date.

Maintenance: This step occurs after installation, and involves making modifications to the system or an individual component to alter attributes or improve performance. These modifications arise either due to change requests initiated by the customer, or defects uncovered during live use of the system. Typically, every change made to the product during the maintenance cycle is recorded and a new product release (called a "maintenance release" and exhibiting an updated revision number) is performed to enable the customer to gain the benefit of the update.

ADVANTAGES

The waterfall model, as described above, offers numerous advantages for software developers. First, the staged development cycle enforces discipline: every phase has a defined start and end point, and progress can be conclusively identified (through the use of milestones) by both vendor and client. The emphasis on requirements and design before writing a single line of code ensures minimal wastage of time and effort and reduces the risk of schedule slippage, or of customer expectations not being met.

Getting the requirements and design out of the way first also improves quality; it's much easier to catch and correct possible flaws at the design stage than at the testing stage, after all the components have been integrated and tracking down specific errors is more complex. Finally, because the first two phases end in the production of a formal specification, the waterfall model can aid efficient knowledge transfer when team members are dispersed in different locations.

CRITICISMS

Despite the seemingly obvious advantages, the waterfall model has come in for a fair share of criticism in recent times. The most prominent criticism revolves around the fact that very often, customers don't really know what they want up-front; rather, what they want emerges out of repeated two-way interactions over the course of the project. In this situation, the waterfall model, with its emphasis on up-front requirements capture and design, is seen as somewhat unrealistic and unsuitable for the vagaries of the real world. Further, given the uncertain nature of customer needs, estimating time and costs with any degree of accuracy (as the model suggests) is often extremely difficult. In general, therefore, the model is recommended for use only in projects which are relatively stable and where customer needs can be clearly identified at an early stage.

Another criticism revolves around the model's implicit assumption that designs can be feasibly translated into real products; this sometimes runs into roadblocks when developers actually begin implementation. Often, designs that look feasible on paper turn out to be expensive or difficult in practice, requiring a re-design and hence destroying the clear distinctions between phases of the traditional waterfall model. Some criticisms also center on the fact that the waterfall model implies a clear division of labor between, say, "designers", "programmers" and "testers"; in reality, such a division of labor in most software firms is neither realistic nor efficient.

DEVELOPMENT LIFE CYCLE PHASES

REQUIREMENTS

The Current System

Data recorded on paper:

The data was initially recorded on sheets of paper, charts for both patients and the doctors' reference. These charts were to be maintained and updated by the doctors' for the patient so that they can be referred in future in case the patient has any other complains. Also the patient had to maintain and carry along with him his record chart whenever he required medical attention so that the doctor can refer to the history of the patient and the prescription he was given previously. This was a very daunting task for both the patient and the doctor.

Handled and archived manually:

All the charts were paper sheets hence handling them and their storage posed a challenge. Many times the charts got destroyed or affected severely due to various factors. Handling many hundreds of charts is a very difficult task due to the number of records itself. Also any person could manipulate or destroy records without authorisation.

How it works?

- Patient is given a chart where all the data is recorded regarding his/her medical history and prescriptions given on the last visit.
- After examination, the doctor fills up the chart, the details of the disease or ailment , symptoms detected and prescribing the relevant drugs.
- The patient is required to maintain the chart and bring it with them on every visit so that the staff as well as the doctor can use it as reference to the history of the patient.
- The patient carries the now updated chart to the pharmacist

- The pharmacist provides the drugs as prescribed
- In case of serious/chronic conditions, the patient is referred to a specialist doctor

Interviews:

- Medical officers were interviewed wherein the details of the current system were acquired.
- Questions were asked about the workings of the current system as to what is the input and output of the system, how the input is processed into the output and who are the people performing these tasks and to what amount of success.
- An outline blueprint was formed so as to accurately evaluate the system's strong points and weak points.
- The weak points which can be improved, of the system were spotted which can be worked upon so as to improve its performance and give the desired results.

The weak points:

- Risk of data loss since it is recorded on paper which can be due to various factors, during storage and handling.
- The whole process is inherently slow and time consuming as it is done manually and might not be accurate and correct at all times.
- Due to the process being manual it is essentially resource intensive as handling hundreds' of records is not a task that can be performed by a single person at all times.
- The records, being unguarded, any staff member can access them or alter them without much risk but can lead to many problems affecting both the staff and the patients and also the reputation of the clinic.

What the client expects

- A system with automatic data processing so as to reduce the manual effort

- A central data location where all information can be stored, allowing patients to visit other hospitals in the same chain without having to carry charts.
- An easy way for new patients to get registered.
- An automatic data backup system so as to secure the data for easier future reference.
- An easy & fast user interface reducing time consumption and effort.
- Reasonable amount of data security to maintain the privacy of the patient and his records for reference.

SPECIFICATION

Functions of the product

- Handle all the patient records
- Automate systems so as to reduce manual efforts
- Make regular backups
- Generate reports and analyse data
- Provide reasonable data security, redundancy and privacy
- Provide an easy & convenient way for new patients to register
- Provide an easy to use User Interface for the staff members to use

How the new system works

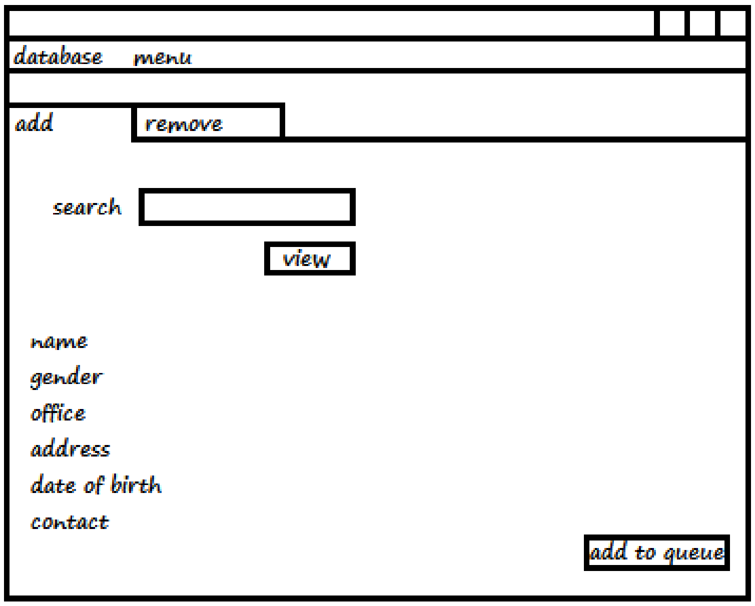
- Instead of a chart, the patient will be given a smart card, eliminating the need of a chart.
- All patient records are stored in a centralized, encrypted, limited access and redundant server for easy and safe access and manipulation of all the records.
- A desktop app will be designed to communicate with the server, handle data entries and display to provide an easy interface to the user.
- A web interface will allow patients to get registered and reduce the time and effort wasted by both the patient and the staff.

Operation in detail

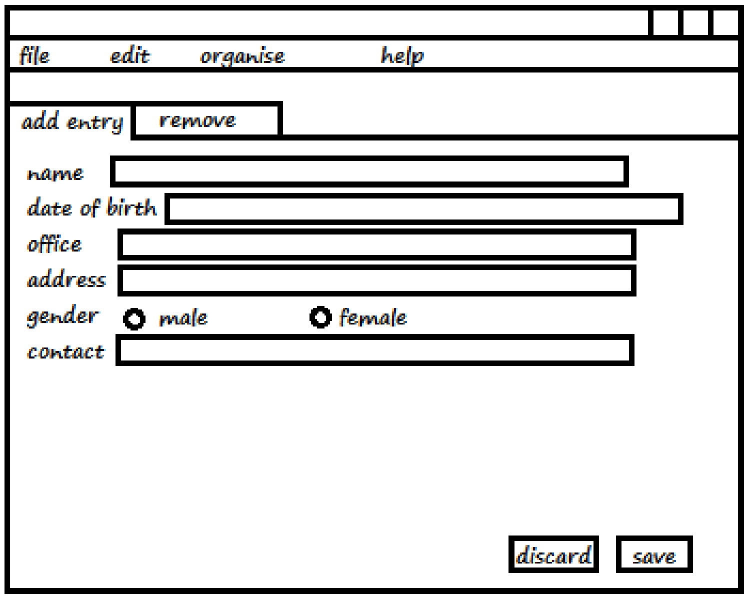
- Patient is given a chart where all the data is recorded regarding his/her medical history and prescriptions given on the last visit.

- After examination, the doctor fills up the chart, the details of the disease or ailment, symptoms detected and prescribing the relevant drugs.
- The patient is required to maintain the chart and bring it with them on every visit so that the staff as well as the doctor can use it as reference to the history of the patient.
- The patient carries the now updated chart to the pharmacist
- The pharmacist provides the drugs as prescribed
- In case of serious/chronic conditions, the patient is referred to a specialist doctor

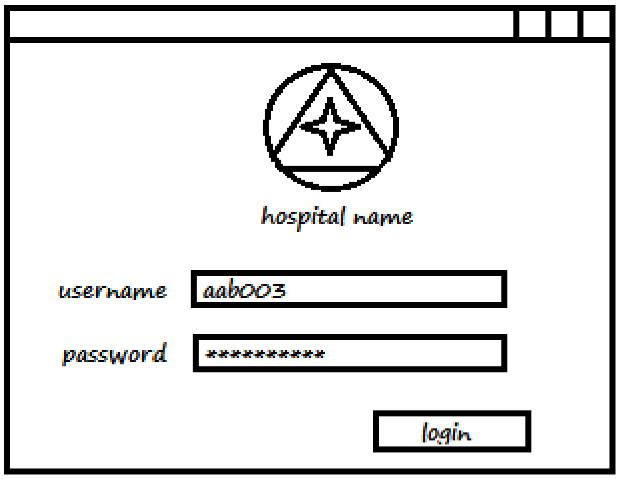
User Interface Mockups



Patient Record Detail Screen



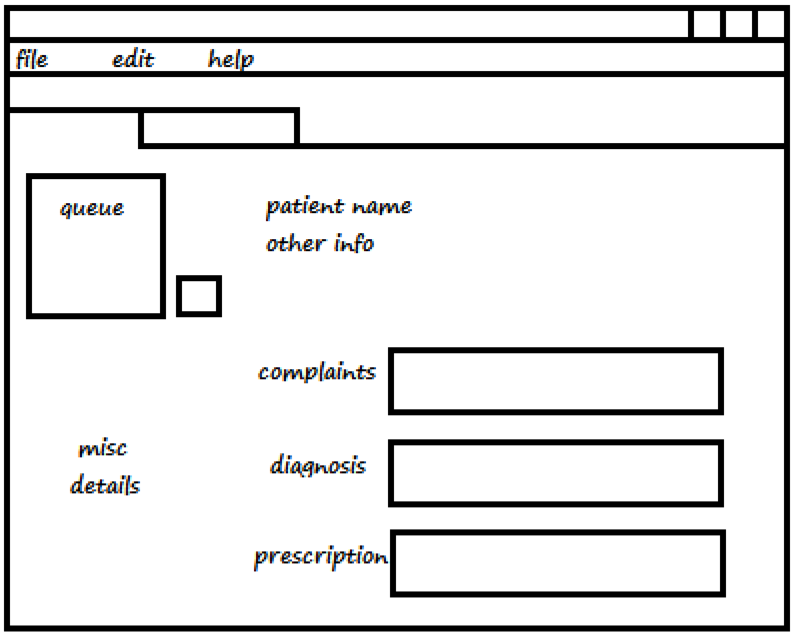
New Patient Record Screen



aab003

****

User Login Dialog Box



Doctor Screen

DESIGN

The Underlying Framework

Frontend

The frontend and the graphical user interface is provided by Microsoft Visual Basic (MS VB) and the .NET Framework.

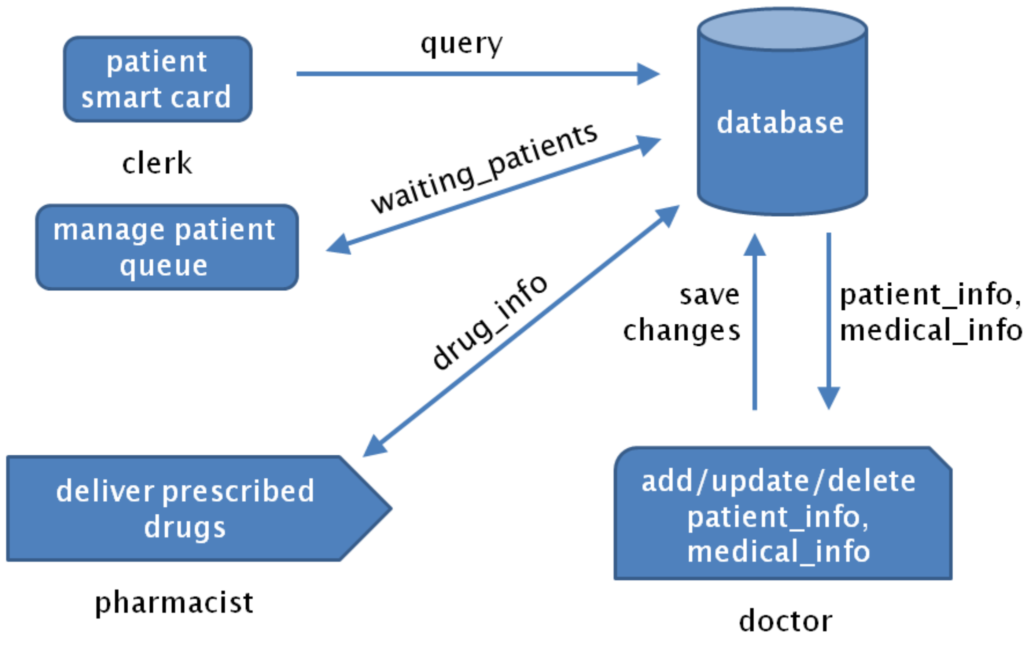
Backend

The app is to be powered by Oracle Database linked with the frontend by ODBC. ODBC is the programming interface that enables applications to access data in Database Management Systems using the Structured Query Language (SQL) as a data access standard.

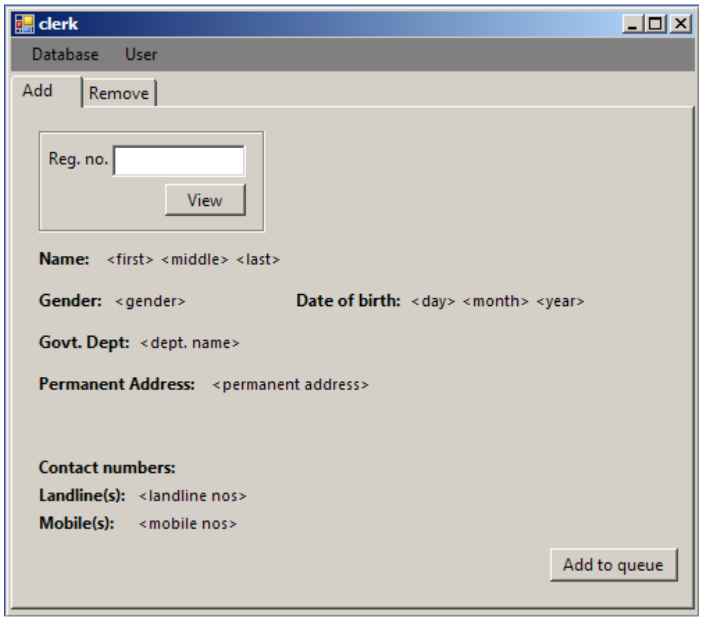
Flow of Data

- Patient is given a chart where all the data is recorded regarding his/her medical history and prescriptions given on the last visit.
- After examination, the doctor fills up the chart, the details of the disease or ailment, symptoms detected and prescribing the relevant drugs.
- The patient is required to maintain the chart and bring it with them on every visit so that the staff as well as the doctor can use it as reference to the history of the patient.
- The patient carries the now updated chart to the pharmacist
- The pharmacist provides the drugs as prescribed
- In case of serious/chronic conditions, the patient is referred to a specialist doctor

Data Flow Diagram



IMPLEMENTATION



Name: <first> <middle> <last>

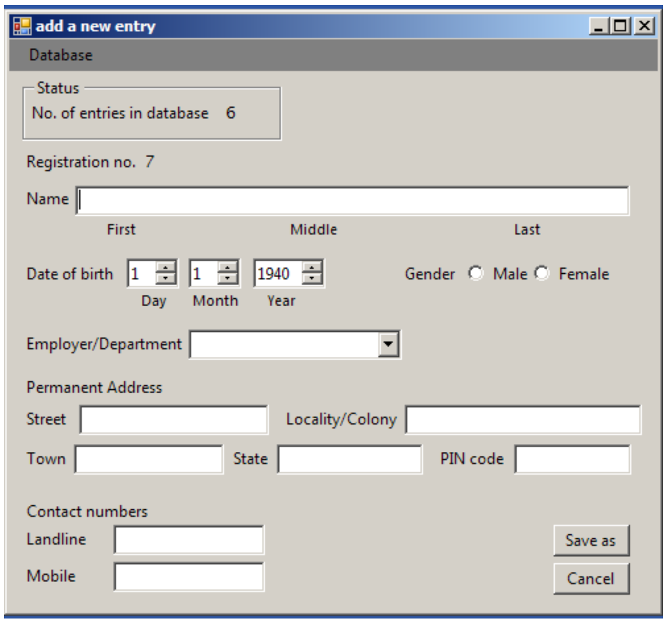
Gender: <gender> **Date of birth:** <day> <month> <year>

Govt. Dept: <dept. name>

Permanent Address: <permanent address>

Contact numbers:
Landline(s): <landline nos>
Mobile(s): <mobile nos>

Patient Record Detail Screen



Status
No. of entries in database 6

Registration no. 7

Name
First Middle Last

Date of birth 1 1 1940
Day Month Year

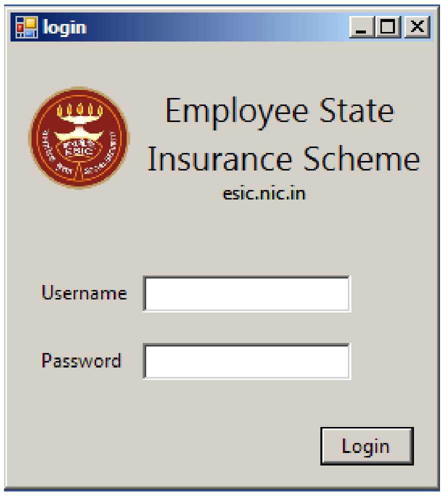
Gender ○ Male ○ Female

Employer/Department

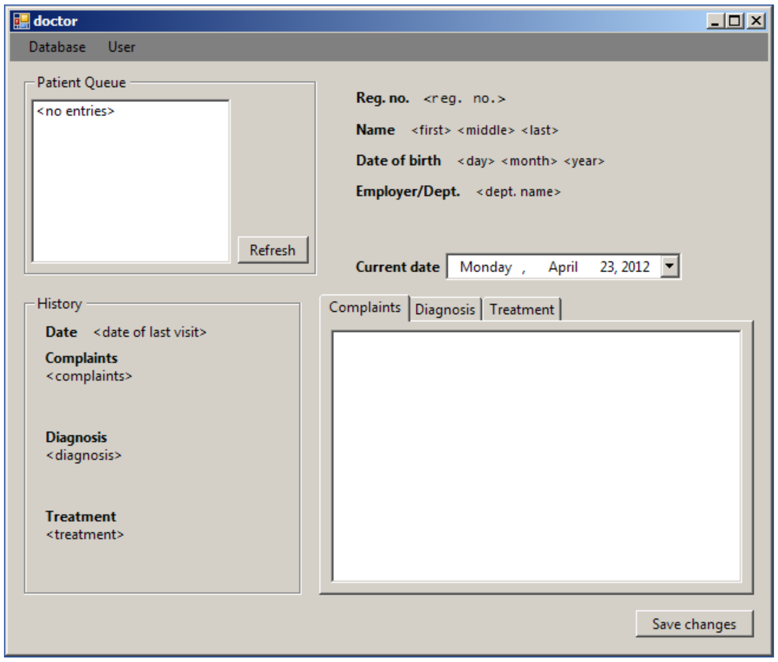
Permanent Address
Street
Locality/Colony
Town
State
PIN code

Contact numbers
Landline
Mobile

New Patient Record Screen

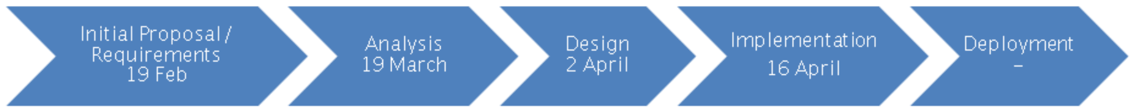


User Login Dialog Box



Doctor Screen

Software Project Management Plan



TASK DESCRIPTION	START DATE	END DATE	DURATION
Requirement Phase	02/12/2013	15/2/2013	4 days
Verification	17/2/2013	19/2/2013	3 days
Specification Phase	20/2/2013	28/3/2013	9 days
Verification	03/01/2013	03/04/2013	4 days
Design Phase	03/05/2013	20/3/2013	15 days
Verification	21/3/2013	25/3/2013	5 days
Implementation Phase	27/3/2013	04/12/2013	16 days
Testing	15/4/2013	19/4/2013	5 days
Integration Phase	29/4/2013	05/05/2013	6 days
Testing	05/06/2013	05/09/2013	4 days

TESTING

Test cases for ***login.vb***

	Entity name	Type	Comments
1.	Username	Text box	Input: alphanumerical
2.	Password	Text box	Input: alphanumerical
3.	Login	Button	Action: valid Username and Password opens relevant forms; if invalid, displays error

	Operation	Expected action	Achieved
1.	Click Login on empty fields	Display error	Yes

Test cases for ***clerk.vb***

	Entity name	Type	Comments
1.	Reg no.	Text box	Input: Numerical
2.	View	Button	Action: valid Reg no. displays associated information; if invalid, displays error
3.	Add to queue	Button	Action: adds current Reg no. to queue
4.	Refresh	Button	Action: updates the queue list
5.	Remove from queue	Button	Action: remove the selected item from queue

	Operation	Expected action	Achieved
1.	Click View on empty Reg. no	Display error	Yes
2.	Enter alphabetical input in Reg. no and click View	Display error	No
3.	Click Remove from queue when Patient queue is empty	Display error	Yes
4.	Add an entry already in queue	Display error	Yes

Test cases for ***new-entry.vb***

	Entity name	Type	Comments
1.	Username	Text box	Input: alphabetical
2.	Date of birth	Numerical up down	Input: numerical
3.	Gender	Radio	Input: Boolean true for male, false for female
4.	Employer	Drop down text box	Input: alphabetical
5.	Street	Text box	Input: alphanumerical
6.	Colony	Text box	Input: alphanumerical
7.	Town	Text box	Input: alphanumerical
8.	State	Text box	Input: alphanumerical
9.	PIN code	Text box	Input: numerical
10.	Landline	Text box	Input: numerical
11.	Mobile	Text box	Input: numerical
12.	Save as	Button	Action: for all valid inputs to above entities, adds an entry to database
13.	Cancel	Button	Action: discards all input

	Operation	Expected action	Achieved
1.	Invalid/incomplete fields when pressing Save as	Display error	Yes

Test cases for ***doctor.vb***

	Entity name	Type	Comments
1.	Patient queue	List box	Displays list of items in queue Action: shows details of selected item
2.	Refresh	Button	Action: updates values in Patient queue
3.	Complaints	Text box	Input: alphabetical
4.	Diagnosis	Text box	Input: alphabetical
5.	Treatment	Text box	Input: alphabetical
6.	Save changes	Button	Action: for all valid inputs to about entities, adds an entry to database

	Operation	Expected action	Achieved
1.	Click Save changes on empty fields	Display error	Yes

Test cases for ***pharma.vb***

	Entity name	Type	Comments
1.	Patient queue	List box	Displays list of entities in queue Action: shows details of selected item
2.	Refresh	Button	Action: updates items in Patient queue

CONCLUSION

The mini-project in the subject of Object-Oriented Software Engineering was done successfully. The final product allows the dispensary staff to handle patient records faster and in a more efficient manner. New patient entries can be added and manipulated with ease.

FURTHER ENHANCEMENTS

The product can be further enhanced in following ways:

- Adding modules to handle pharmacy medicine stock.
- Designing an online interface that would let patients apply for registrations remotely.
- Adding modules to display photographs of the patients to enhance identification.
- Designing a hardware subsystem that lets patients swipe the ID cards, thus eliminating the need of clerk module entirely.
- Adding an extensible subsystem that allows linking with other management systems (for instance, with a specialist hospital).