**CHAPTER 1**

**INTRODUCTION**

The popularity of online shopping is growing day by day. According to an ACNielsen study conducted in 2005, one-tenth of the world’s population is shopping online. Germany and Great Britain have the largest number of online shoppers, and UPI Transaction is the most popular mode of payment (59 percent). About 350 million transactions per year were reportedly carried out by Barclaycard, the largest UPI Transaction company in the United Kingdom, toward the end of the last century. Retailers like Wal-Mart typically handle much larger number of UPI Transaction transactions including online and regular purchases. As the number of UPI Transaction users rises world-wide, the opportunities for attackers to steal UPI Transaction details and, subsequently, commit fraud are also increasing. The total UPI Transaction fraud in the United States itself is reported to be $2.7 billion in 2005 and estimated to be $3.0 billion in 2006, out of which $1.6 billion and $1.7 billion, respectively, are the estimates of online fraud.

Credit-card-based purchases can be categorized into two types:

1) Physical card and

2) Virtual card.

In a physical-card-based purchase, the cardholder presents his card physically to a merchant for making a payment. To carry out fraudulent transactions in this kind of purchase, an attacker has to steal the UPI Transaction. If the cardholder does not realize the loss of card, it can lead to a substantial financial loss to the UPI Transaction company. In these cond. kind of purchase, only some important information about a card (card number, expiration date, secure code) is required to make the payment. Such purchases are normally done on the Internet or over the telephone. To commit fraud in these types of purchases, a fraudster simply needs to know the card details. Most of the time, the genuine cardholder is not aware that someone else has seen or stolen his card information. The only way to detect this kind of fraud is to analyze the spending patterns on every card and to figure out any inconsistency with respect to the “usual” spending patterns. Fraud detection based on the analysis of existing purchase data of cardholder is a promising way to reduce the rate of successful UPI Transaction frauds. Since humans tend to exhibit specific behaviouristic profiles, every cardholder can be represented by a set of patterns containing information about the typical purchase category, the time since the last purchase, the amount of money spent, etc. Deviation from such patterns is a potential threat to the system. Several techniques for the detection of UPI Transaction fraud have been proposed in the last few years.

* 1. **Literature Survey:-**

**UPI Transaction Fraud Detection:-**

UPI Transaction fraud detection has drawn a lot of research interest and a number of techniques, with special emphasis on data mining and neural networks, have been suggested. Ghosh and Reilly have proposed UPI Transaction fraud detection with a neural network. They have built a detection system, which is trained on a large sample of labelled UPI Transaction account transactions. These transactions contain exam-ple fraud cases due to lost cards, stolen cards, application fraud, counterfeit fraud, mail-order fraud, and nonreceived issue (NRI) fraud. Recently, Syeda et al. have used parallel granular neural networks (PGNNs) for improving the speed of data mining and knowledge discovery process in UPI Transaction fraud detection. A complete system has been implemented for this purpose. Stolfo et al. suggest a UPI Transaction fraud detection system (FDS) using Metalearning techniques to learn models of fraudulent UPI Transaction transactions. Metalearning is a general strategy that provides a means for combining and integrating a number of separately built classifiers or models. A metaclassifier is thus trained on the correlation of the predictions of the base classifiers. The same group has also worked on a cost-based model for fraud and intrusion detection. They use Python agents for Metalearning (JAM), which is a distributed data mining system for UPI Transaction fraud detection. A number of important performance metrics like True Positive—False Positive (TP-FP) spread and accuracy have been defined by them. Aleskerov et al. present CARDWATCH, a database mining system used for UPI Transaction fraud detection. The system, based on a neural learning module, provides an interface to a variety of commercial databases. Kim and Kim have identified skewed distribution of data and mix of legitimate and fraudulent transactions as the two main reasons for the complexity of UPI Transaction fraud detection. Based on this observation, they use fraud density of real transaction data as a confidence value and generate the weighted fraud score to reduce the number of misdetections. Fan et al. suggest the application of distributed data mining in UPI Transaction fraud detection. Brause et al. have developed an approach that involves advanced data mining techniques and neural network algorithms to obtain high fraud coverage. Chiu and Tsai have proposed Web services and data mining techniques to establish a collaborative scheme for fraud detection in the banking industry. With this scheme, participating banks share knowledge about the fraud patterns in a heterogeneous and distributed environment. To establish a smooth channel of data exchange, Web services techniques such as XML, SOAP, and WSDL are used. Phua et al. have done an extensive survey of existing data-mining-based FDSs and published a comprehensive report. Prodromidis and Stolfo use an agent-based approach with distributed learning for detect-ing frauds in UPI Transaction transactions. It is based on artificial intelligence and combines inductive learning algorithms and Metalearning methods for achieving higher accuracy. Phua et al. suggest the use of metaclassifier similar to in fraud detection problems. They consider naive Bayesian C4.5, and Back Propagation neural networks as the base classifiers. A metaclassifier is used to determine which classifier should be considered based on skewness of data. Although they do not directly use UPI Transaction fraud detection as the target application, their approach is quite generic. Vatsa et al. have recently proposed a game-theoretic approach to UPI Transaction fraud detection. They model the interaction between an attacker and an FDS as a multistage game between two players, each trying to maximize his payoff. The problem with most of the abovementioned approaches is that they require labelled data for both genuine, as well as fraudulent transactions, to train the classifiers. Getting real-world fraud data is one of the biggest problems associated with UPI Transaction fraud detection. Also, these approaches cannot detect new kinds of frauds for which labelled data is not available. In contrast, we present a Hidden Markov Model (AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING)-based UPI Transaction FDS, which does not require fraud signatures and yet is able to detect frauds by considering a cardholder’s spending habit. We model a UPI Transaction transaction processing sequence by the stochastic process of an AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. The details of items purchased in individual transactions are usually not known to an FDS running at the bank that issues UPI Transactions to the cardholders. This can be represented as the underlying finite Markov chain, which is not observable. The transactions can only be observed through the other stochastic process that produces the sequence of the amount of money spent in each transaction. Hence, we feel that AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING is an ideal choice for addressing this problem. Another important advantage of the AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING-based approach is a drastic reduction in the number of False Positives (FPs)—transactions identified as malicious by an FDS although they are actually genuine. Since the number of genuine transactions is a few orders of magnitude higher than the number of malicious transactions, an FDS should be designed in such a way that the number of FPs is as low as possible. Otherwise, due to the “base rate fallacy” effect, bank administrators may tend to ignore the alarms. To the best of our knowledge, there is no other published literature on the application of AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING for UPI Transaction fraud detection.

**AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING Background:-**

An AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING is a double embedded stochastic process with two hierarchy levels. It can be used to model much more complicated stochastic processes as compared to a traditional Markov model. An AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING has a finite set of states governed by a set of transition probabilities. In a particular state, an outcome or observation can be generated according to an associated probability distribution. It is only the outcome and not the state that is visible to an external observer. AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING-based applications are common in various areas such as speech recognition, bioinformatics, and genomics. In recent years, Joshi and Phoba have investigated the capabilities of AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING in anomaly detection. They classify TCP network traffic as an attack or normal using AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. Cho and Park suggest an AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING-based intrusion detection system that improves the modelling time and performance by considering only the privilege transition flows based on the domain knowledge of attacks. Ourston et al. have proposed the application of AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING in detecting multistage network attacks. Hoang et al.present a new method to process sequences of system calls for anomaly detection using AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. The key idea is to build a multilayer model of program behaviours based on both AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERINGs and enumerating methods for anomaly detection. Lane has used AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING to model human behaviour. Once human behaviour is correctly modelled, any detected deviation is a cause for concern since an attacker is not expected to have behaviour similar to the genuine user. Hence, an alarm is raised in case of any deviation. An AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING can be characterized by the following:-

1. N is the number of states in the model. We denote the set of states S= {S1, S2...SN}, where Si, i = 1,2...N is an individual state. The state at time instant t is denoted by qt.

2. M is the number of distinct observation symbols per state. The observation symbols correspond to the physical output of the system being modelled. We denote the set of symbols V= {V1, V2,...VM}, where Vi = 1,2,...M is an individual symbol.

* 1. **Existing System:-**
* The card holder faced a lot of trouble before the investigation finish.
* All the transaction is maintained in a log, we need to maintain a huge data.
* We don't know the person how is using the card online, we just capture the ip address for verification purpose.
* So there need a help from the cyber crime to investigate the fraud.
  1. **Problem Definition:-**

In case of the existing system the fraud is detected after the fraud is done that is, the fraud is detected after the complaint of the holder. And so the card holder faced a lot of trouble before the investigation finish. And also as all the transaction is maintained in a log, we need to maintain a huge data, and also now a day's lot of online purchase are made so we don't know the person how is using the card online, we just capture the ip address for verification purpose. So there need a help from the cyber crime to investigate the fraud. To avoid the entire above disadvantage we propose the system to detect the fraud in a best easy way.

* 1. **Proposed System:-**

In this system ,we present a hidden morkov model(AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING) Which does not required fraud signatures and yet is able to detect frauds by considering a cardholder's spending habit. Card transaction processing sequence by the stochastic process of an AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. The details of items purchased in individual transactions are usually not known to an FDS running at the bank that issues UPI Transactions to the cardholder. Hence, we feel that AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING is an ideal choice for addressing this problem. Another important advantage of the AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING based approach is a drastic reduction is the number reduction in the number of false positives transactions identified a malicious by an FDS although they are actually genuine . An FDS runs at a UPI Transaction issuing bank. Each incoming transaction is submitted to the FDS for verification. FDS receives the card details and the values if purchases to verify, whether the transaction is genuine or not. The types of goods that are bought in that transaction are not known to the FDS. It tries to find nay anomaly in the transaction based on the spending profile of the cardholder, shipping address, shipping address, and billing addresses. 

**Advantages :-**

* The detection of the fraud use of the card is found much faster than existing system.
* No need check the original user as we maintain a log .
* The log which is maintained will also be a proof for the bank for the transaction made.
* We can find the most accurate detection using this technique.
* This reduce the tedious work of an employee in bank.

**CHAPTER 2**

**PROJECT PLANNING AND MANAGEMENT**

**2.1 Objective :-**

Software project management is an activity of organising, planning and scheduling the software projects. The goal of software project management is to deliver the software product in given time and within the budget. It is also necessary that the software project should project should be developed in accordance with the requirements of the organisation. The software project managers are responsible for planning and scheduling the software projects. Managing software projects is difficult because of Following reasons:-

1. Other engineering products are visible. But the software is intangible. It is not a physical entity that can be touched. Hence software project managers can see the progress of software project with the help of documents only.
2. The large projects are different in nature. Hence even the managers who have adequate experiences of previous projects may find difficulties in the current project.
3. There are no standard software process. It changes from one organisation to another.

Because of these difficulties the most of the software projects could not meet the budgets or fail to deliver on time.

**2.2 Scope :-**

* The detection of the fraud use of the card is found much faster than existing system.
* No need check the original user as we maintain a log .
* The log which is maintained will also be a proof for the bank for the transaction made.
* We can find the most accurate detection using this technique.
* This reduce the tedious work of an employee in bank.

**2.3 Cost Estimation :-**

Software cost estimation is an important factor in software project. Large cost estimation errors can affect profit and cost of software project. If the project cost exceeds then it will seriously affect the project. But software cost and effort estimation will never be exact. The cost and effort estimation can be done in following ways :-

* Delay the estimation as far as possible. This is not a feasible way of estimation. Because then we have to wait for long period to get project estimation.
* Use the estimates of earlier similar projects. If the current project is similar to earlier project then past efforts business conditions and deadlines are equivalent. But sometimes past efforts are not good indicators.
* Some decomposition techniques can be used to generate project cost and effort estimations.

The decomposition techniques are nothing but the “divide and conquer” techniques used for project estimation. That means decompose the project into major functions. Perform cost and effort estimations for these functionalities and related software activities.Thus for each major function, software cost and effort estimation can be obtained in step by step manner.

* Use of some empirical model for project cost eatimation.

The empirical cost estimation model is used for determining the cost of the project based on observations and experiences. The estimation can be made in the form of

E = f(Vi)

Where E is effort or cost or duration and f(V i) is the function for V i project estimation in terms of LOC or function point(fp).

There is another way of estimating a cost and that is use of automated estimation tools.

There are some systems available in which automated tools along with Graphical User Interface(GUI) are used to determine the cost of the project.

* 1. **COCOMO Model:-**

COCOMO is one of the most widely used software estimation models in the world. This model is developed in 1981 by Barry Boehm to give an estimate of the number of man-months it will take to develop a software product. COCOMO predicts the efforts and schedule of a software product based on size of the software. COCOMO stands for “Constructive Cost Model”.

COCOMO has three different models that reflect the complexity

* Basic model
* Intermediate model
* Detailed model

Similarly there are three classes of software projects.

* 1. **Organic mode :-** In this mode, relatively small, simple software projects with a small team are handled. Such a team should have good application experiences to less rigid requirements.
  2. **Semi-detached Projects :-** In this class an intermidiate projects in which teams with mixed experience level are handled. Such projects may have mix of rigid and less than rigid requirements.
  3. **Embedded Projects :-** In this class, projects with tight hardware, software and operational constraints are handled.

Let us understand each model in detail.

1. Basic Model :- The basic COCOMO model estimates the software development effort using only Lines Of Code(LOC). Various equations in this model are –

E = ab (KLOC)bb

D = Cb(E)db

P = E/D

Where E is the effort applied in person- months.

D is the development time in chronological months.

KLOC means kilo line of code for the project.

P is the total number of persons required to accomplish the project.

The coefficient ab, bb, cb, db  for three modes are as given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software Projects | ab | bb | Cb | db |
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi- detached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

Table :- The coefficient for three modes

**Merits of Basic COCOMO Model :-**

1. Basic COCOMO model is good for quick, early, rough order of magnitude estimate of software project.

**Limitations of Basic Model :-**

1. The accuracy of this model is limited because it does not consider certain factors for cost estimation of software. These factors are hardware constraints, personal quality, and experience, modern techniques and tools.
2. The estimate of COCOMO model are within a factor of 1.3 only 29% of the time and within the factor of 2 only 60% of time.

**CHAPTER 3**

**SOFTWARE ENGINEERING PARADIGM**

Process models are proposed in order to adopt the systematic approach in the software development. These models define the distinct set of activities, tasks and work products that are required to create high quality software. These process models are also called as prescriptive process model, generic software process models or software paradigms.

The process model can be defined as an abstract representation of process. The process model is chosen based on nature of software project.

* 1. **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 republishedwith a foreword by Benington pointing out that the process was not in fact performed in strict top-down, 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.

RR

Coding

Maintenance

Testing

Design

Requirement Gathering and Analysis

Figure 3.1:- Waterfall Model

**Advantages :-**

1. Waterfall model is simple and easy to implement.
2. In this model, output is generated after each stage, therefore it has high visibility. The client and project manager gets a feel that there is considerable progress.
3. Project management, both at internal level and client’s level, is easy  
   again because of visible outputs after each. Deadlines can be set for  
   the completion of each phase and evolution can be done from time to time, to check if project is going as per milestones.
4. This methodology is significantly better than the haphazard approach to develop software.

**Disadvantages :-**

1) It is difficult to follow the sequential flow in software devlopment process. If some changes are made at some phase then it may cause some confusion.

2) The requirement analysis is done initially, and sometimes it is not possible to state all the requirements explicitly in the beginning. This causes difficulty in the project.

3) The customer can see the working model of the project only at the end. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.

4) Linear nature of waterfall model induces blocking states, because certain tasks may be dependent on some previous tasks. Hence it is necessary to accomplish all the dependent tasks first. It may cause long waiting time.

* 1. **RAD Model :-**

Rapid application development (RAD) is a software development methodology that uses minimal planning in favour of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

Rapid application development is a software development methodology that involves methods like iterative development and software prototyping. According to Whitten (2004), it is a merger of various structured techniques, especially data-driven Information Engineering, with prototyping techniques to accelerate software systems development.

In rapid application development, structured techniques and prototyping are especially used to define users' requirements and to design the final system. The development process starts with the development of preliminary data models and business process models using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems".

RAD approaches may entail compromises in functionality and performance in exchange for enabling faster development and facilitating application maintenance.

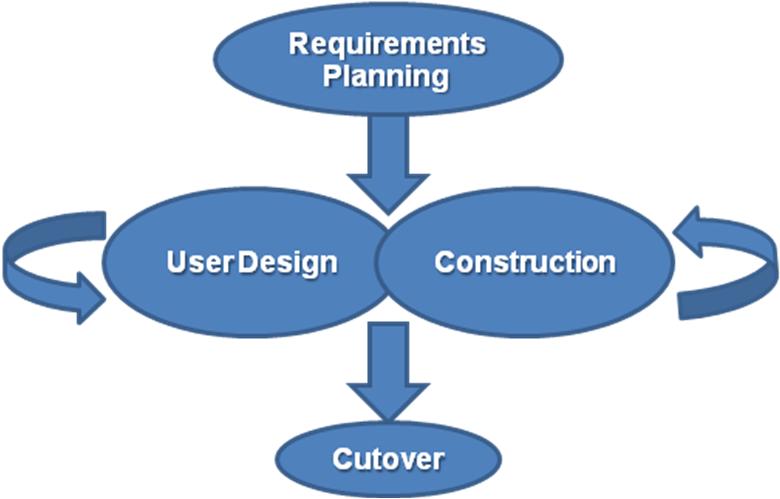


Figure 3.2 :- RAD Model In software Engineering

**Advantages :**

1. Quick initial reviews are possible.
2. Constant integration isolate problems and incourage customer feedback.

**Disadvantages :**

1. Required a systematic approach for moduralise.
2. Required highly skilled and well trained developers.
   1. **Spiral Model :-**

The spiral model is a [software development process](http://en.wikipedia.org/wiki/Software_development_process) combining elements of both [design](http://en.wikipedia.org/wiki/Design) and [prototyping](http://en.wikipedia.org/wiki/Prototyping)-in-stages, in an effort to combine advantages of [top-down and bottom-up](http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design) concepts. Also known as the spiral lifecycle model (or spiral development), it is a systems development method (SDM) used in [information technology](http://en.wikipedia.org/wiki/Information_technology) (IT). This model of development combines the features of the prototyping and the [waterfall model](http://en.wikipedia.org/wiki/Waterfall_model). The spiral model is intended for large, expensive and complicated projects.

The spiral model is divided into a number of framework activities. These framework activities are denoted by task regions.

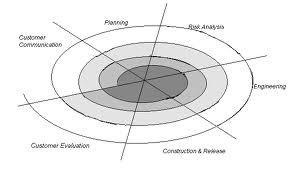


Figure 3.3:- Spiral Model in software Engineering

**Advantages :-**

1. Requirement changes can be made at every stage.
2. Risks can be identified and rectified before they get problematic.

**Disadvantages :-**

1. It is based on customer communication. If the communication is not proper then the software product that gets developed will not be up to the mark.
2. It demands considerable risk assessment. If the risk assessment is done properly then only the successful product can be obtained.
   1. **Incremental Model :-**

The incremental Model is an evolution of the waterfall model, where the waterfall model is incrementally applied.

The series of releases is referred to as “increments”, with each increment providing more functionality to the customers. After the first increment, a core product is delivered, which can already be used by the customer. Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly. This process continues, with increments being delivered until the complete product is delivered. The incremental philosophy is also used in the agile process model.

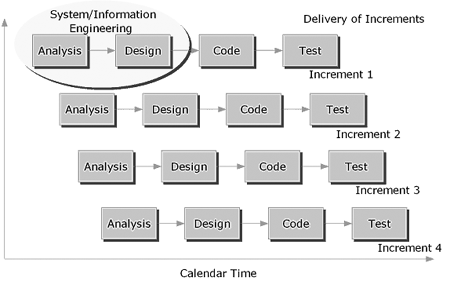
****

Figure 3.4 :- Incremental Model in software engineering

**Advantages :-**

1. After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
2. It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration. This allows for more targeted and rigorous testing of each element within the overall product.

**Disadvantages :-**

1. Resulting cost may exceed the cost of the organization.
2. As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.
   1. **Verification and Validation Model :-**

The V-model represents a software development process (also applicable to hardware development) which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

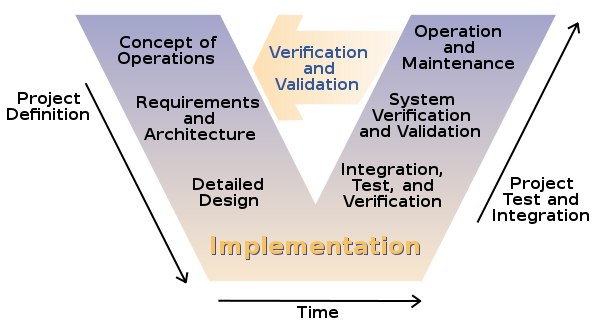


Figure 3.5 :- Verification and Validation Model in software Engineering

**Advantages :-**

1. Proactive defect tracking i.e defects r found at early stages even may be in the development phase before [application](http://www.geekinterview.com/question_details/29763) is tested.

2. Avoids the downward flow of the defect

3. Reduces the cost for fixing the defect since defects will be found in early stages

4. It is a fast method.

**Disadvantages :-**

1.More peoples are requires to work.

**3.6 Prototyping Model :-**

In prototyping model initially the requirement gathering is done. Developer and customer define overall objectives; identify areas needing more requirement gathering. Then a quick design is prepared. This design represents what will be visible to user-in input and output format. From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements. When working prototype is built, developer use existing program fragments or program generates to throw away the prototype and rebuilt the system to high quality.

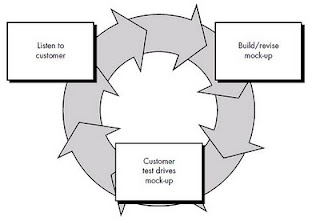
[](http://3.bp.blogspot.com/_sxBFFOfnkJw/S6oZ-Q8dabI/AAAAAAAAAJg/saZPB5Q_znE/s1600/model.JPG)

Figure 3.6 :- Incremental model in software engineering

**Advantages of prototyping:**

1. Reduced time and costs
2. Improved and increased user involvement.

**Disadvantages of prototyping:**

1. Insufficient analysis.
2. User confusion of prototype and finished system
3. Developer misunderstanding of user objectives
4. Expense and time of implementing prototyping

As far as our project is concern, we are using the **spiral model** of our project. This model is best fit for our project. It provides our project is more flexibility than any other model and good work flow.

**The selection of this model is because of the following advantages :-**

**Spiral Model :-**

**Advantages :-**

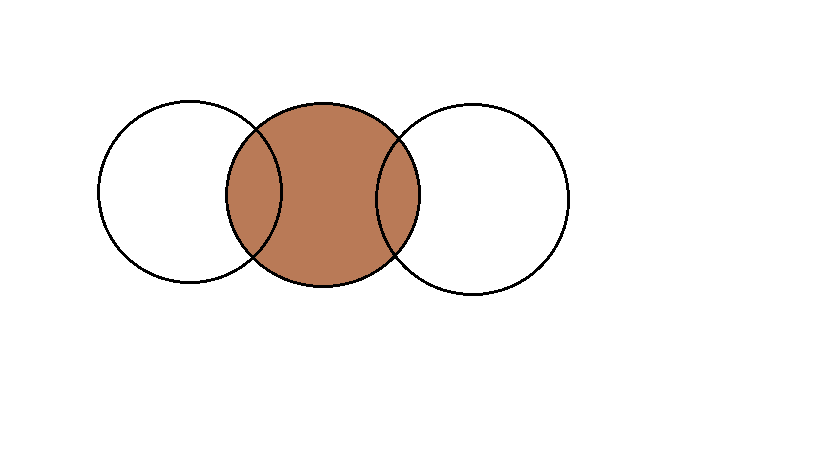
* 1. Spiral Life Cycle Model is one of the most flexible SDLC models in place. Development phases can be determined by the project manager, according to the complexity of the project.
  2. [Project monitoring](http://www.ianswer4u.com/2011/12/earned-value-management-analysis.html) is very easy and effective. Each phase, as well as each loop, requires a review from concerned people. This makes the model more transparent.
  3. Risk management is one of the in-built features of the model, which makes it extra attractive compared to other models.
  4. Changes can be introduced later in the life cycle as well. And coping with these changes isn’t a very big headache for the project manager.
  5. It is suitable for high risk projects, where business needs may be unstable.
  6. A highly customized product can be developed using this.

**CHAPTER 4**

**REQUIREMENT ANALYSIS**

**Introduction :-**

Requirement analysis is an intermediate phase between system engineering and software design.

 System Requirement Software

Engineering analysis design

Figure 4.1:- Requirement analysis : an intermediate step

Requirement analysis produces a software specification.

**How is requirement analysis helpful ?**

**Analyst –** The requirement analysis help the ‘analyst’ to refine software allocation. Using requirement analysis various models such as data model, functional model and behavioral model can be defined.

**Designer –** After requirement analysis, the designer can design for data, architectural interface and component level designs.

**Developer –** Using requirements specification and design the software can be developed.

1. **Functional Requirements:-**

**Modules**

1. New card
2. Login
3. Security information
4. Transaction
5. Verification

**Module Description**

**New card**

In this module, the customer gives there information to enroll a new card. The information is all about there contact details. They can create there own login and password for there future use of the card.

**Login**

In Login Form module presents site visitors with a form with username and password fields. If the user enters a valid username/password combination they will be granted access to additional resources on website. Which additional resources they will have access to can be configured separately.

**Security information**

In Security information module it will get the information detail and its store’s in database. If the card lost then the Security information module form arise. It has a set of question where the user has to answer the correctly to move to the transaction section. It contain informational privacy and informational self-determination are addressed squarely by the invention affording persons and entities a trusted means to user, secure, search, process, and exchange personal and/or confidential information.

**Transaction**

The method and apparatus for pre-authorizing transactions includes providing a communications device to a vendor and a UPI Transaction owner. The UPI Transaction owner initiates a UPI Transaction transaction by communicating to a UPI Transaction number, and storing therein, a distinguishing piece of information that characterizes a specific transaction to be made by an authorized user of the UPI Transaction at a later time. The information is accepted as "network data" in the data base only if a correct personal identification code (PIC) is used with the communication. The "network data" will serve to later authorize that specific transaction. The UPI Transaction owner or other authorized user can then only make that specific transaction with the UPI Transaction. Because the transaction is pre-authorized, the vendor does not need to see or transmit a PIC.

**Verification**

Verification information is provided with respect to a transaction between an initiating party and a verification-seeking party, the verification information being given by a third, verifying party, based on confidential information in the possession of the initiating party. In verification the process will seeks card number and if the card number is correct the relevant process will be executed. If the number is wrong, mail will be sent to the user saying the card no has been block and he can’t do the further transaction.

**2. Non- Functional Requirements**

**Performance Requirements**

* The system responds to each user input within few seconds.

**Security requirements**

* Security is implemented in our project as we have implemented any security things like login form or authentication to users. As our system which is secured so it cannot be accessed by any unauthorized user.
* **Software Quality Attributes**

1. Reliability

The system will never crash or hang, other than as the result of an operating system error. The system shall provide graceful degradation in the face of Frauds.

1. Maintainability

The code will be fully documented. Each function will be commented with pre- and post-conditions the code will be modular to permit future modifications.

3. Portability

The software will be designed to run on the following platforms:

* Microsoft Windows XP, Vista, Win 7

**CHAPTER 5**

**MINIMUM HARDWARE AND SOFTWARE REQUIREMENT**

**5.1 Hardware Required:-**

The following are the Hardware requirement for the project.

* Processor : Pentium 4
* RAM : 4GB or more
* Hard disk : 16 GB or more

**5.2 Software Tool Required:-**

The following are the software required for the project.

**5.2.1 JDK (Python Development Kit)**

The JDK is a development environment for building application, applets, and components using the Python programming language. The JDK include tools useful for developing and testing programs written in the Python programming language and running on the Python platform.

Contents of JDK :-

* Development Tools
* Runtime Environment
* Additional Libraries

**5.2.2 Windows Support:-**

* Windows XP SP -2/SP- 3/VISTA/Windows Seven

**CHAPTER 6**

**FEASIBILITY STUDY**

Feasibility study is the process of determination of whether or not a project is worth doing. Feasibility studies are undertaken within tight time constraints and normally culminate in a written and oral feasibility report. I have taken two weeks in feasibility study with my co-developer. The contents and recommendations of this feasibility study helped us as a sound basis for deciding how to precede the project. It helped in taking decisions such as which software to use, hardware combinations, etc.

1. Technical Feasibility.

2. Economical Feasibility.

3. Social Feasibility.

**1. Technical Feasibility:-**

Technical feasibility determines whether the work for the project can be done with the existing equipment, software technology and available personal. Technical feasibility is concerned with specifying equipment and software that will satisfy the user requirement.

This project is feasible on technical remarks also, as the proposed system is more beneficiary in terms of having a sound proof system with new technical components installed on the system. The proposed system can run on any machines supporting Windows and Internet services and works on the best software and hardware that had been used while designing the system so it would be feasible in all technical terms of feasibility.

**2. Economical Feasibility:-**

Economical feasibility determines whether there are sufficient benefits in creating to make the cost acceptable, or is the cost of the system too high. As this signifies cost-benefit analysis and savings. On the behalf of the cost-benefit analysis, the proposed system is feasible and is economical regarding its pre-assumed cost for making a system.

We classified the costs of Transport according to the phase in which they occur. As we know that the system development costs are usually one-time costs that will not recur after the project has been completed. For calculating the Development costs we evaluated certain cost categories viz.

1. Personal costs

2. Computer usage

3. Supply and equipments costs

4. Cost of any new computer equipments and software.

**3. 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.

**CHAPTER 7**

**DESIGN CONCEPT AND PRINCIPLES**

**7.1 Definition:-**

Software design is a process of problem solving and planning for a [software](http://en.wikipedia.org/wiki/Software) solution. After the purpose and specifications of software are determined, [software developers](http://en.wikipedia.org/wiki/Software_developer) will [design](http://en.wikipedia.org/wiki/Design) or employ [designers](http://en.wikipedia.org/wiki/Designer) to develop a plan for a solution. It includes low-level component and [algorithm](http://en.wikipedia.org/wiki/Algorithm) implementation issues as well as the [architectural](http://en.wikipedia.org/wiki/Software_architecture) view.

Following issues are considered while designing the software.

1. **Abstraction :-**

At each stage of software design process levels of abstraction should be applied to refine the software solution. At the higher level of abstraction, the solution should be start in broad terms and in the lower level more detailed description of the solution is given.

1. **Modularity :-**

* The software is divided in to separately named an addressable components that called as modules. Creating such a modules bring the modularity in software.
* Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:-

1. Modular decomposability
2. Modular composability
3. Modular understandibilty
4. Modular continuity
5. Modular protection
6. **Refinement :-**

* Refinement is actually a process of elaboration.
* Stepwise refinement is a top-down design strategies proposed by Niklaus WIRTH.
* The architecture of a program is developed by successively refining levels of procedural detail.
* The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
* Abstraction and refinement are complementary concepts. The major difference is that – in the abstraction low-level details are suppressed.

Refinement helps the designer to elaborate low-level details.

**7.2** **Design** **Principles:-**

There are following design principles for the any software:

* **The design process should not suffer from “tunnel vision”:**

A good designer should consider alternative approaches, judging each based on the requirement of the problem, the resources available to do the job.

* **The design should be traceable to the analysis model**.

Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.

* **The design should not reinvent the wheel:**

Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be choosen as an alternative to reinvention. Time is short and resources are limited. Design time should be invested in representing truly new ideas and integrating those pattern those already exist.

* **The design should “minimize the intellectual distance” between the software and the problem in the real world:**

That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.

* **The design should exhibit uniformity and integration:**

A design is uniform if it appears that one person develop the entire thing. Rules of style and format should be define for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

* **The design should be structured to accommodate change:**

The design concept discussed in the next section enable a design to achieve this principle.

* **The design should be structured to degrade gently, even when aberrant data, event, or operating conditions are encountered:**

Well designed software should never “bomb”. It should be design to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.

* **Design is not coding and coding is not design:**

Even when detail procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decision made at the coding level address the small implementation details that enable the procedural designed to be coded.

**7.3 Fundamental of Design:-**

A set of fundamental design concepts has evolved. They are:

1. [**Abstraction**](http://en.wikipedia.org/wiki/Abstraction) - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
2. [**Refinement**](http://en.wikipedia.org/wiki/Program_refinement) - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.
3. [**Modularity**](http://en.wikipedia.org/wiki/Modularity) - Software architecture is divided into components called modules.
4. [**Software Architecture**](http://en.wikipedia.org/wiki/Software_Architecture) - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. A good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.
5. [**Control Hierarchy**](http://en.wikipedia.org/w/index.php?title=Control_Hierarchy&action=edit&redlink=1)- A program structure that represents the organization of a program component and implies a hierarchy of control.
6. [**Structural Partitioning**](http://en.wikipedia.org/w/index.php?title=Structural_Partitioning&action=edit&redlink=1) - The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.
7. [**Data Structure**](http://en.wikipedia.org/wiki/Data_Structure) - It is a representation of the logical relationship among individual elements of data.
8. [**Software Procedure**](http://en.wikipedia.org/w/index.php?title=Software_Procedure&action=edit&redlink=1) - It focuses on the processing of each modules individually
9. [**Information Hiding**](http://en.wikipedia.org/wiki/Information_Hiding) - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information

**7.4 Design Methods / Types:-**

The elements of design model are :-

1. Data Design
2. Architectural Design
3. Interface Design
4. Component-level Design
5. **Data Design :-**

The data design is used to transform the information domain model of analysis phase into the data structures. These data structures play an important role in software implementations. The entity relationship diagram and data dictionary are used to create the data design model. In entity relationship diagram the relationships among the data objects is defined and in data dictionary detailed data contents are given, Hence ERD and data dictionary are used to implement the data design.

1. **Architectural Design :-**

The architectural design is used to represent the relationship between major structural elements with the help of some “design patterns”. Hence data flow diagrams from analysis model serve as the basis for architectural design.

1. **Interface Design :-**

The ‘interface design’ describes how software interacts within itself. An interface means flow of information and specific type of behaviour. Hence by using the data flow and control flow diagrams the interface design can be modelled.

1. **Component-level Design :-**

In the ‘component-level design’ the structural elements of software architecture into procedural description of software components. Hence the component-level design can be obtained using State Transition Diagrams (STD), Control Specification (CSPEC) and Process Specification (PSPEC).

**Module Diagram :-**

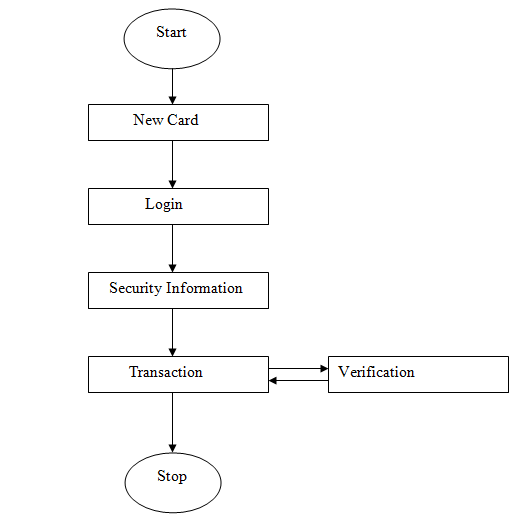
****

Figure 7.1 Module diagram

**E-R Diagram :-**

User

Account History

HAS

Transaction

HAS

HAS

Verification

Figure 7.2 E-R Diagram

**UML Diagrams :-**



Figure 7.3 Use case diagram



Figure 7.4 :- Class Diagram



Figure 7.5 :- Sequence Diagram

****

Fig 7.6.- Activity Diagram



Fig:–7.7 Collaboration Diagram

**CHAPTER 8**

**CODING**

**8.1 :-CODING FOR CUSTOMER REGISTRATION MODULE**

import Python.sql.Connection;

import Python.sql.DriverManager;

import Python.sql.Statement;

import Python.util.Random;

import Pythonx.swing.JOptionPane;

public class Customer\_Details extends Pythonx.swing.JFrame

{

public Customer\_Details()

{

initComponents();

}

GEN-BEGIN:initComponents

private void initComponents() {

bfr=new StringBuffer();

jPanel1 = new Pythonx.swing.JPanel();

jPanel2 = new Pythonx.swing.JPanel();

jLabel2 = new Pythonx.swing.JLabel();

jLabel15 = new Pythonx.swing.JLabel();

First = new Pythonx.swing.JTextField();

jLabel4 = new Pythonx.swing.JLabel();

Bank = new Pythonx.swing.JComboBox();

jLabel5 = new Pythonx.swing.JLabel();

Card = new Pythonx.swing.JComboBox();

jLabel7 = new Pythonx.swing.JLabel();

Account = new Pythonx.swing.JTextField();

jLabel9 = new Pythonx.swing.JLabel();

Dob = new Pythonx.swing.JTextField();

jLabel11 = new Pythonx.swing.JLabel();

Questions = new Pythonx.swing.JComboBox();

Last = new Pythonx.swing.JTextField();

jLabel3 = new Pythonx.swing.JLabel();

jLabel6 = new Pythonx.swing.JLabel();

Salary = new Pythonx.swing.JTextField();

Range = new Pythonx.swing.JLabel();

jLabel8 = new Pythonx.swing.JLabel();

jLabel10 = new Pythonx.swing.JLabel();

Email = new Pythonx.swing.JTextField();

jLabel12 = new Pythonx.swing.JLabel();

Answer = new Pythonx.swing.JTextField();

Submit = new Pythonx.swing.JButton();

Next = new Pythonx.swing.JButton();

jLabel13 = new Pythonx.swing.JLabel();

jLabel14 = new Pythonx.swing.JLabel();

Number = new Pythonx.swing.JLabel();

Password = new Pythonx.swing.JLabel();

jLabel1 = new Pythonx.swing.JLabel();

setDefaultCloseOperation(Pythonx.swing.WindowConstants.EXIT\_ON\_CLOSE);

jPanel1.setBackground(new Python.awt.Color(255, 102, 102));

jPanel1.setLayout(null);

jPanel2.setBackground(new Python.awt.Color(255, 153, 153));

jLabel2.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel2.setText("First Name:");

jLabel4.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel4.setText("Bank Name:");

Bank.setFont(new Python.awt.Font("Palatino Linotype", 1, 14));

Bank.setModel(new Pythonx.swing.DefaultComboBoxModel(new String[] {"SELECT", "SBI", "BOM", "INDIAN", "AXIS" }));

Bank.addActionListener(new Python.awt.event.ActionListener() {

public void actionPerformed(Python.awt.event.ActionEvent evt) {

CardActionPerformed(evt);

}

});

jLabel5.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel5.setText("Type of cards:");

Card.setFont(new Python.awt.Font("Palatino Linotype", 1, 14));

Card.setModel(new Pythonx.swing.DefaultComboBoxModel(new String[] {"SELECT", "PLATINUM", "MASTER CARD", "VISA", "MAESTRO" }));

Card.addActionListener(new Python.awt.event.ActionListener() {

public void actionPerformed(Python.awt.event.ActionEvent evt) {

CardActionPerformed(evt);

}

});

jLabel7.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel7.setText("A/C No:");

jLabel9.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel9.setText("Dob:(d/m/yy)");

jLabel11.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel11.setText("Questions:");

Questions.setFont(new Python.awt.Font("Palatino Linotype", 1, 14));

Questions.setModel(new Pythonx.swing.DefaultComboBoxModel(new String[] { "Favourite Film?", "Academic Concern?", "Favourite City?" }));

Questions.addActionListener(new Python.awt.event.ActionListener()

{

public void actionPerformed(Python.awt.event.ActionEvent evt)

{

QuestionsActionPerformed(evt);

}

});

jLabel3.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel3.setText("Last Name:");

jLabel6.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel6.setText("Salary:");

jLabel8.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel8.setText("Your Range:");

jLabel10.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel10.setText("E\_mail:");

jLabel12.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel12.setText("Answers:");

Submit.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

Submit.setText("Submit");

Submit.addActionListener(new Python.awt.event.ActionListener()

{

public void actionPerformed(Python.awt.event.ActionEvent evt)

{

SubmitActionPerformed(evt);

}

});

Next.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

Next.setText("Clear");

Next.addActionListener(new Python.awt.event.ActionListener()

{

public void actionPerformed(Python.awt.event.ActionEvent evt)

{

NextActionPerformed(evt);

}

});

First.addKeyListener(new Python.awt.event.KeyListener()

{

public void keyReleased(Python.awt.event.KeyEvent ke)

{

}

public void keyPressed(Python.awt.event.KeyEvent ke)

{

char ch=ke.getKeyChar();

if(Character.isDigit(ch))

{

JOptionPane.showMessageDialog(null,"Digit not allowed");

First.setText("");

}

}

public void keyTyped(Python.awt.event.KeyEvent ke)

{

char ch=ke.getKeyChar();

if(Character.isDigit(ch))

{

JOptionPane.showMessageDialog(null,"Digit not allowed");

First.setText("");

}

}

});

Last.addKeyListener(new Python.awt.event.KeyListener()

{

public void keyReleased(Python.awt.event.KeyEvent ke)

{

}

public void keyPressed(Python.awt.event.KeyEvent ke)

{

char ch=ke.getKeyChar();

if(Character.isDigit(ch))

{

JOptionPane.showMessageDialog(null,"Digit not allowed");

Last.setText("");

}

}

public void keyTyped(Python.awt.event.KeyEvent ke)

{

char ch=ke.getKeyChar();

if(Character.isDigit(ch))

{

JOptionPane.showMessageDialog(null,"Digit not allowed");

First.setText("");

}

}

});

jLabel13.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel13.setText("Card Number");

jLabel14.setFont(new Python.awt.Font("Palatino Linotype", 1, 18));

jLabel14.setText("Password");

Pythonx.swing.GroupLayout jPanel2Layout = new Pythonx.swing.GroupLayout(jPanel2);

jPanel2.setLayout(jPanel2Layout);

jPanel2Layout.setHorizontalGroup( jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(185, 185, 185)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addComponent(jLabel5, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(Card, 0, 174, Short.MAX\_VALUE))

.addGroup(jPanel2Layout.createSequentialGroup() .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel2, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel4, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED) .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING)

.addComponent(Bank, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 139, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(First, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 139, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(jPanel2Layout.createSequentialGroup() .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jLabel9, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel11, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel7, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING)

.addComponent(Dob, Pythonx.swing.GroupLayout.Alignment.LEADING, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 174, Short.MAX\_VALUE)

.addComponent(Questions, Pythonx.swing.GroupLayout.Alignment.LEADING, 0, 174, Short.MAX\_VALUE)

.addComponent(Account, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 174, Short.MAX\_VALUE)))

.addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel14, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 140, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel13)) .addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED) .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(Number)

.addComponent(Password, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 163, Short.MAX\_VALUE))

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))) .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(77, 77, 77) .addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(133, 133, 133)

.addComponent(Email, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 143, Short.MAX\_VALUE))

.addComponent(jLabel10, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(jPanel2Layout.createSequentialGroup()

.addComponent(jLabel6, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(Salary, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 143, Short.MAX\_VALUE))

.addGroup(jPanel2Layout.createSequentialGroup()

.addComponent(jLabel3, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(Last, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 143, Short.MAX\_VALUE))

.addGroup(jPanel2Layout.createSequentialGroup()

.addComponent(jLabel8, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(Range, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 143, Short.MAX\_VALUE))

.addGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()

.addGap(5, 5, 5)

.addComponent(jLabel12, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 129, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(Pythonx.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(Answer, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 138, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)))

.addGap(122, 122, 122))

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(56, 56, 56)

.addComponent(Submit, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 134, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(55, 55, 55)

.addComponent(Next, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 89, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap())))

);

jPanel2Layout.setVerticalGroup jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(39, 39, 39)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel2, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(First, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel3, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Last, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(26, 26, 26)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel4, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Bank, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Salary, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel6, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(35, 35, 35)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel5, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Card, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel8, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Range, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGap(32, 32, 32)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel10, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel7, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Account, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)))

.addComponent(Email, Pythonx.swing.GroupLayout.Alignment.TRAILING, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(30, 30, 30)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Answer, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Dob, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel9, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(31, 31, 31))

.addGroup(jPanel2Layout.createSequentialGroup()

.addComponent(jLabel12, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)))

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Questions, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel11, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(45, 45, 45)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Number, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 30, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel13, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 28, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(30, 30, 30)

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel14, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 27, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Password, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(27, Short.MAX\_VALUE))

.addGroup(Pythonx.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.BASELINE)

.addComponent(Submit, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 33, Pythonx.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(Next, Pythonx.swing.GroupLayout.PREFERRED\_SIZE, 33, Pythonx.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(47, 47, 47))))

);

jPanel1.add(jPanel2);

jPanel2.setBounds(-60, 40, 960, 520);

jLabel1.setFont(new Python.awt.Font("Arial Black", 1, 24)); // NOI18N

jLabel1.setForeground(new Python.awt.Color(51, 0,255));

jLabel1.setText("User\_Details");

jPanel1.add(jLabel1);

jLabel1.setBounds(310, 0, 260, 35);

Pythonx.swing.GroupLayout layout = new Pythonx.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addComponent(jPanel1, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 897, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

layout.createParallelGroup(Pythonx.swing.GroupLayout.Alignment.LEADING)

.addComponent(jPanel1, Pythonx.swing.GroupLayout.DEFAULT\_SIZE, 614, Short.MAX\_VALUE)

);

pack();

}// </editor-fold>//GEN-END:initComponents

private void CardActionPerformed(Python.awt.event.ActionEvent evt)

{//GEN-FIRST:event\_CardActionPerformed

String d=(String) Card.getSelectedItem();

if(d.equals("PLATINUM"))

{

Range.setText("20000");

}

else if(d.equals("MASTER CARD"))

{

Range.setText("30000");

} else if(d.equals("VISA"))

{

Range.setText("40000");

} else

if(d.equals("MAESTRO"))

{

Range.setText("50000");

}

Random r=new Random();

int num=r.nextInt(1000000);

System.out.println(num);

String s = new Integer(num).toString();

Number.setText(s);

final int PASSWORD\_LENGTH = 8;

StringBuffer sb = new StringBuffer();

for (int x = 0; x < PASSWORD\_LENGTH; x++)

{

sb.append((char)((int)(Math.random()\*26)+97));

}

System.out.println(sb.toString());

Password.setText(sb.toString());

}//GEN-LAST:event\_CardActionPerformed

private void SubmitActionPerformed(Python.awt.event.ActionEvent evt) {//GEN-FIRST:event\_SubmitActionPerformed

int b=Bank.getSelectedIndex();

int ct=Card.getSelectedIndex();

if((b!=0) && (ct!=0) && (!First.getText().equals("")) && (!Last.getText().equals("")) && (!Account.getText().equals("")) && (!Salary.getText().equals("")) && (!Range.getText().equals("")) && (!Dob.getText().equals("")) && (!Email.getText().equals("")) && (!Answer.getText().equals("")))

{

Connection conn=null;

Statement s=null;

int limit=Integer.parseInt(Range.getText());

System.out.println("\n\nLimit is := "+limit);

int lower=limit\*20/100;

int middle=limit\*50/100;

int upper=limit\*100/100;

String ss="INSERT INTO Customer\_Details("+"First\_Name,"+"Last\_Name,"+"Bank\_Name,"+"Salary,"+"Card\_Type,"+"Range,"+"AC\_No,"+"DOB,"+"E\_MAIL,"+"QUESTIONS,"+"ANSWERS,"+"CARD\_NUMBER,"+"PASSWORD,"+"lower\_range,"+"middle\_range,"+"upper\_range)"

+"VALUES('"+First.getText()+"','"+Last.getText()+"','"+ Bank.getSelectedItem()+"','"+Salary.getText()+"','"+Card.getSelectedItem()+"',"+Integer.parseInt(Range.getText())+",'"+Account.getText()+"','"+Dob.getText()+"','"+Email.getText()+"','"+Questions.getSelectedItem()+"','"+Answer.getText()+

"','"+Number.getText()+"','"+Password.getText()+"',"+lower+","+middle+","+upper+")";

try {

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

conn=DriverManager.getConnection("jdbc:odbc:Credit");

// JOptionPane.showMessageDialog(null,"Connection Established");

s=conn.createStatement();

s.executeUpdate(ss);

JOptionPane.showMessageDialog(null, "Data Successfully Inserted");

conn.close();

this.dispose();

}

catch (Exception e)

{

System.out.println(e);

JOptionPane.showMessageDialog(null,e);

}

First.setText("");

Last.setText("");

Bank.setSelectedIndex(0);

Card.setSelectedIndex(0);

Dob.setText("");

Email.setText("");

Questions.setSelectedIndex(0);

Salary.setText("");

Password.setText("");

Range.setText("");

Number.setText("");

Answer.setText("");

Account.setText("");

}

else

{

JOptionPane.showMessageDialog(null,"Fill complete Details ");

}

}

private void NextActionPerformed(Python.awt.event.ActionEvent evt)

{

First.setText("");

Last.setText("");

Bank.setSelectedIndex(0);

Card.setSelectedIndex(0);

Dob.setText("");

Email.setText("");

Questions.setSelectedIndex(0);

Salary.setText("");

Password.setText("");

Range.setText("");

Number.setText("");

Answer.setText("");

Account.setText("");

}

private void QuestionsActionPerformed(Python.awt.event.ActionEvent evt) {//GEN-FIRST:event\_QuestionsActionPerformed

// TODO add your handling code here:

}//GEN-LAST:event\_QuestionsActionPerformed

public static void main(String args[])

{

new Customer\_Details().setVisible(true);

}

GEN-BEGIN:variables

private Pythonx.swing.JTextField Account;

private Pythonx.swing.JTextField Answer;

private Pythonx.swing.JComboBox Bank;

private Pythonx.swing.JComboBox Card;

private Pythonx.swing.JTextField Dob;

private Pythonx.swing.JTextField Email;

private Pythonx.swing.JTextField First;

private Pythonx.swing.JTextField Last;

private Pythonx.swing.JButton Next;

private Pythonx.swing.JLabel Number;

private Pythonx.swing.JLabel Password;

private Pythonx.swing.JComboBox Questions;

private Pythonx.swing.JLabel Range;

private Pythonx.swing.JTextField Salary;

private Pythonx.swing.JButton Submit;

private Pythonx.swing.JLabel jLabel10;

private Pythonx.swing.JLabel jLabel11;

private Pythonx.swing.JLabel jLabel12;

private Pythonx.swing.JLabel jLabel13;

private Pythonx.swing.JLabel jLabel14;

private Pythonx.swing.JLabel jLabel15;

private Pythonx.swing.JLabel jLabel2;

private Pythonx.swing.JLabel jLabel3;

private Pythonx.swing.JLabel jLabel4;

private Pythonx.swing.JLabel jLabel5;

private Pythonx.swing.JLabel jLabel6;

private Pythonx.swing.JLabel jLabel7;

private Pythonx.swing.JLabel jLabel8;

private Pythonx.swing.JLabel jLabel9;

private Pythonx.swing.JPanel jPanel1;

private Pythonx.swing.JPanel jPanel2;

}

**CHAPTER 9**

**SOFTWARE TESTING**

**9.1 Software Testing :**

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding [software bugs](http://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

* meets the requirements that guided its design and development,
* works as expected,
* can be implemented with the same characteristics,
* and satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the [Agile](http://en.wikipedia.org/wiki/Agile_software_development) approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as [Agile](http://en.wikipedia.org/wiki/Agile_software_development), often employ [test-driven development](http://en.wikipedia.org/wiki/Test-driven_development) and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

Testing can never completely identify all the defects within software. Instead, it furnishes a criticism or comparison that compares the state and behaviour of the product against [oracles](http://en.wikipedia.org/wiki/Oracle_(software_testing))—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, [contracts](http://en.wikipedia.org/wiki/Design_by_Contract), comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. Software testing is the process of attempting to make this assessment.

**9.2 Need For Testing :**

Testing wasessential for the following reasons :-

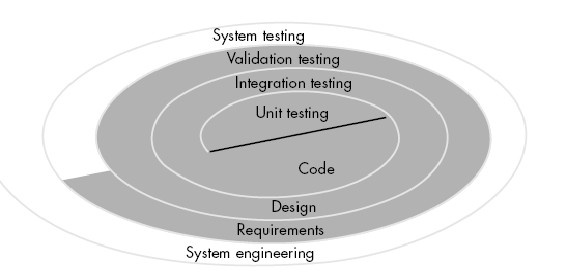
* Existence of program detects of inadequacies.
* A software behaviour as intended by its designer.
* Conformance with requirement s specification/user needs.
* Assess the operational reliability of the system.
* Reflect the frequency of actual user inputs.
* Find the faults, which causes the output anomaly.
* Checks for detects flaws and deficiencies in the requirements.
* Checks whether the software is operationally useful.
* Excercise the program using data like the real data processed by the program.

**9.3 Testing Strategies :**

The philosophy behind testing is to find errors. Test cases are devised with this purpose in mind. Test case is a set of data that a system will process as nominal input.

**Characteristics of Good Test :**

* Test are likely to catch bugs.
* No redundancy
* Not to Simple or too complex.

****

**Fig.: Testing Strategies**

**Unit Testing :-**

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch [corner cases](http://en.wikipedia.org/wiki/Corner_case) or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Depending on the organization's expectations for software development, unit testing might include [static code analysis](http://en.wikipedia.org/wiki/Static_code_analysis), data flow analysis metrics analysis, peer code reviews, code coverage analysis and other software verification practices.

**Integration Testing :-**

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

**Validation Testing :-**

At the culmination of integration testing, software is completely assembled as a pack-age, interfacing errors have been uncovered and corrected, and a final series of soft-ware tests—validation testing—may begin. Validation can be defined in many ways, but a simple (albeit harsh) definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer. At this point a battle-hardened software developer might protest: "Who or what is the arbiter of reasonable expectations?"

**Alpha testing :-**

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

**Beta testing :-**

Beta testing comes after alpha testing and can be considered a form of external [user acceptance testing](http://en.wikipedia.org/wiki/User_acceptance_testing). Versions of the software, known as [beta versions](http://en.wikipedia.org/wiki/Beta_version), are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or [bugs](http://en.wikipedia.org/wiki/Computer_bug). Sometimes, beta versions are made available to the open public to increase the [feedback](http://en.wikipedia.org/wiki/Feedback#In_organizations) field to a maximal number of future users.

**White-Box Testing :-**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. [in-circuit testing](http://en.wikipedia.org/wiki/In-circuit_test) (ICT).

While white-box testing can be applied at the [unit](http://en.wikipedia.org/wiki/Unit_testing), [integration](http://en.wikipedia.org/wiki/Integration_testing) and [system](http://en.wikipedia.org/wiki/System_testing) levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

* [API](http://en.wikipedia.org/wiki/Application_programming_interface) testing (application programming interface) – testing of the application using public and private APIs
* [Code coverage](http://en.wikipedia.org/wiki/Code_coverage) – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
* [Fault injection](http://en.wikipedia.org/wiki/Fault_injection) methods – intentionally introducing faults to gauge the efficacy of testing strategies
* [Mutation testing](http://en.wikipedia.org/wiki/Mutation_testing) methods
* [Static testing](http://en.wikipedia.org/wiki/Static_testing) methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important [function points](http://en.wikipedia.org/wiki/Function_points) have been tested. Code coverage as a [software metric](http://en.wikipedia.org/wiki/Software_metric) can be reported as a percentage for:

* Function coverage, which reports on functions executed
* Statement coverage, which reports on the number of lines executed to complete the test.

100% statement coverage ensures that all code paths, or branches (in terms of [control flow](http://en.wikipedia.org/wiki/Control_flow)) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

**Black-box Testing :-**

[http://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/Blackbox.svg/200px-Blackbox.svg.png](http://en.wikipedia.org/wiki/File:Blackbox.svg)

[http://bits.wikimedia.org/static-1.22wmf1/skins/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:Blackbox.svg)

**Black box diagram**

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it.

Black-box testing methods include: [**equivalence partitioning**](http://en.wikipedia.org/wiki/Equivalence_partitioning)**,**[**boundary value analysis**](http://en.wikipedia.org/wiki/Boundary_value_analysis)**,**[**all-pairs testing**](http://en.wikipedia.org/wiki/All-pairs_testing)**,**[**state transition tables**](http://en.wikipedia.org/wiki/State_transition_table)**,**[**decision table**](http://en.wikipedia.org/wiki/Decision_table)**testing,**[**fuzz testing**](http://en.wikipedia.org/wiki/Fuzz_testing)**,**[**model-based testing**](http://en.wikipedia.org/wiki/Model-based_testing)**,**[**use case**](http://en.wikipedia.org/wiki/Use_case)**testing,**[**exploratory testing**](http://en.wikipedia.org/wiki/Exploratory_testing)**and specification-based testing.**

Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough [test cases](http://en.wikipedia.org/wiki/Test_case) to be provided to the tester, who then can simply verify that for a given input, the output value (or behaviour), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be [functional](http://en.wikipedia.org/wiki/Functional_testing) or [non-functional](http://en.wikipedia.org/wiki/Non-functional_testing), though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight." Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: [unit](http://en.wikipedia.org/wiki/Unit_test), [integration](http://en.wikipedia.org/wiki/Integration_testing), [system](http://en.wikipedia.org/wiki/System_testing) and [acceptance](http://en.wikipedia.org/wiki/Acceptance_test). It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

**CHAPTER 10**

**MAINTENANCE**

Software maintenance in [software engineering](http://en.wikipedia.org/wiki/Software_engineering) is the modification of a software product after delivery to correct faults, to improve performance or other attributes.

A common perception of maintenance is that it merely involves fixing [defects](http://en.wikipedia.org/wiki/Software_bug). However, one study indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions (Pigosky 1997). This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the [system](http://en.wikipedia.org/wiki/Software_maintenance).

Software maintenance and [evolution](http://en.wikipedia.org/wiki/Software_evolution) of systems was first addressed by [Meir M. Lehman](http://en.wikipedia.org/wiki/Meir_M._Lehman) in 1969. Over a period of twenty years, his research led to the formulation of [Lehman's Laws](http://en.wikipedia.org/wiki/Software_evolution#Lehman.27s_Laws_of_Software_Evolution) (Lehman 1997). Key findings of his research include that maintenance is really evolutionary development and that maintenance decisions are aided by understanding what happens to systems (and software) over time. Lehman demonstrated that systems continue to evolve over time. As they evolve, they grow more complex unless some action such as [code refactoring](http://en.wikipedia.org/wiki/Code_refactoring) is taken to reduce the complexity.

The key software maintenance issues are both managerial and technical. Key management issues are: alignment with customer priorities, staffing, which organization does maintenance, estimating costs. Key technical issues are: limited understanding, [impact analysis](http://en.wikipedia.org/wiki/Change_impact_analysis), testing, and maintainability measurement. Four types are encountered during the support phase :

**Correction:**

Even with the best quality assurance activities, it is likely that the customer will uncover defect in the software. Corrective maintenance changes the software to correct defects.

**Adaptation:**

Over time, the original environment (e.g. CPU operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance result in modification to the software to accommodate change to its external environment.

**Enhancement:**

As software is used, the customer/user will recognise additional function that will provide benefit. The perfective maintenance the extend the software beyond its original functional requirement.

**Prevention:**

Computer software deteriorates due to change, and because of this, preventive maintenance, often called software reengineering must be conducted to enable the software to serve the need of its end user. In essence, preventive maintenance makes changes to computer program so that they can be more easily corrected, adapted, and enhanced.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

In this project, we have proposed an application of AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING in UPI Transaction fraud detection. The different steps in UPI Transaction transaction processing are represented as the underlying stochastic process of an AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. We have used the ranges of transaction amount as the observation symbols, whereas the types of item have been considered to be states of the AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING. We have suggested a method for finding the spending profile of cardholders, as well as application of this knowledge in deciding the value of observation symbols and initial estimate of the model parameters. It has also been explained how the AUTO ENCODER, LOCAL OUTLIER FACTOR, KMEANS CLUSTERING can detect whether an incoming transaction is fraudulent or not. Experimental results show the performance and effectiveness of our system and demonstrate the usefulness of learning the spending profile of the cardholders. Comparative studies reveal that the Accuracy of the system is close to 80 percent over a wide variation in the input data. The system is also scalable for handling large volumes of transactions.

Don’t allow your business to fall victim to fraudulent UPI Transaction charges. Our fully customizable Fraud Detection Suite (FDS) helps you to find and eliminate harmful transactions. When suspicious activity is detected, you have the control over whether to approve, decline or hold it for further review. The E-Complish Fraud Detection Suite easily integrates with other solutions for check and UPI Transaction payments using the VirtualPay system, our all-in-one online payments console.

**FUTURE SCOPE :-**

* Detailed reporting on suspicious activity
* Option to hold questionable transactions for review
* Ability to block transactions from historically fraudulent IPs
* Robust searching feature allows you to quickly locate transactions
* Choose from pre-drafted customer responses, or customize your own
* Receive instant notification of questionable activity directly to your inbox
* Customizable filters for easy identification of fraudulent charges, including but not limited to:
  + Set minimum and maximum amounts
  + Monitor amount of charges applied to a card per hour
  + Scan for address discrepancies
  + Locate abnormal activity from a single IP address, and block IPs known to be fraudulent

## BENEFITS:-

* Decreased fees and other costs associated with UPI Transaction fraud
* Increased confidence in legitimacy of transactions
* Settings tailored to your business necessities
* Advanced technology to take your UPI Transaction security to the next level
* We based intuitive design is quick to learn and easy to use
* Seamlessly integrates with other E-Complish solutions using the online payments system.

**CHAPTER 12**

**REFERENCES**

[1] “Global Consumer Attitude Towards On-Line Shopping,”

<http://www2.acnielsen.com/reports/documents/2005_cc_online> shopping.pdf, Mar. 2007.

[2] D.J. Hand, G. Blunt, M.G. Kelly, and N.M. Adams, “Data Mining for Fun and Profit,” Statistical Science, vol. 15, no. 2, pp. 111-131, 2000.

[3] “Statistics for General and On-Line Card Fraud,” <http://www>. epaynews.com/statistics/fraud.html, Mar. 2007.

[4] S. Ghosh and D.L. Reilly, “UPI Transaction Fraud Detection with a Neural-Network,” Proc. 27th Hawaii Int’l Conf. System Sciences: Information Systems: Decision Support and Knowledge-Based Systems, vol. 3, pp. 621-630, 1994.

[5] M. Syeda, Y.Q. Zhang, and Y. Pan, “Parallel Granular Networks for Fast UPI Transaction Fraud Detection,” Proc. IEEE Int’l Conf. Fuzzy Systems, pp. 572-577, 2002.

[6] S.J. Stolfo, D.W. Fan, W. Lee, A.L. Prodromidis, and P.K. Chan, “UPI Transaction Fraud Detection Using Meta-Learning: Issues and Initial Results,” Proc. AAAI Workshop AI Methods in Fraud and Risk Management, pp. 83-90, 1997.

[7] S.J. Stolfo, D.W. Fan, W. Lee, A. Prodromidis, and P.K. Chan, “Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project,” Proc. DARPA Information Survivability Conf. and Exposition, vol. 2, pp. 130-144, 2000.

[8] E. Aleskerov, B. Freisleben, and B. Rao, “CARDWATCH: A Neural Network Based Database Mining System for UPI Transaction Fraud Detection,” Proc. IEEE/IAFE: Computational Intelligence for Financial Eng., pp. 220-226, 1997.

[9] M.J. Kim and T.S. Kim, “A Neural Classifier with Fraud Density Map for Effective UPI Transaction Fraud Detection,” Proc. Int’l Conf. Intelligent Data Eng. and Automated Learning, pp. 378-383, 2002.

[10] C. Chiu and C. Tsai, “A Web Services-Based Collaborative Scheme for UPI Transaction Fraud Detection,” Proc. IEEE Int’l Conf. e-Technology, e-Commerce and e-Service, pp. 177-181, 2004.

[11] V. Vatsa, S. Sural, and A.K. Majumdar, “A Game-theoretic Approach to UPI Transaction Fraud Detection,” Proc. First Int’l Conf. Information Systems Security, pp. 263-276, 2005.

[12] S.S. Joshi and V.V. Phoha, “Investigating Hidden Markov Models Capabilities in Anomaly Detection,” Proc. 43rd ACM Ann. Southeast Regional Conf., vol. 1, pp. 98-103, 2005.

[13] S.B. Cho and H.J. Park, “Efficient Anomaly Detection by Modeling Privilege Flows Using Hidden Markov Model,” Computer and Security, vol. 22, no. 1, pp. 45-55, 2003.

[14] D. Ourston, S. Matzner, W. Stump, and B. Hopkins, “Applications of Hidden Markov Models to Detecting Multi-Stage Network Attacks,” Proc. 36th Ann. Hawaii Int’l Conf. System Sciences, vol. 9, pp. 334-344, 2003.

[15] T. Lane, “Hidden Markov Models for Human/Computer Interface Modeling,” Proc. Int’l Joint Conf. Aicial Intelligence, Workshop Learning about Users, pp. 35-44, 1999.