# Software Engineering Project Report

**Report:**   
This report shall be submitted in three steps:  
      1. Part 1 (Section 1 Customer Problem Statement and Section 2 System Requirements)  
      2. Part 2 (Section 3 Functional Requirements Specification and Section 4 User Interface Specification)  
      3. Entire Report



**PREPARED BY-**

**DHAVAL MANIYAR**

**DIVYANSH BAMANKAR**

**DHEERAJ PATEL**

**Problem statement : -**

In this digital world, everyone is busy in there day to day life, so there is a big task for contractors, builders, etc. to find the workers because no one has time to find the worker according to their requirements. The same problem happens with the worker also who wants to work but due to the absence of communication with the person (who wants to hire workers), he/she does not get any work easily.

* The challenge of finding skilled workers continues to be one of the biggest concerns among contractors.
* One of the most direct means to address this issue is attracting more people to work, but the industry has struggled to do so and this was also time consuming.
* The workers who want to work but it may not get the opportunity.
* The trust issues are also there for the completion of the work.

**Solution : -**

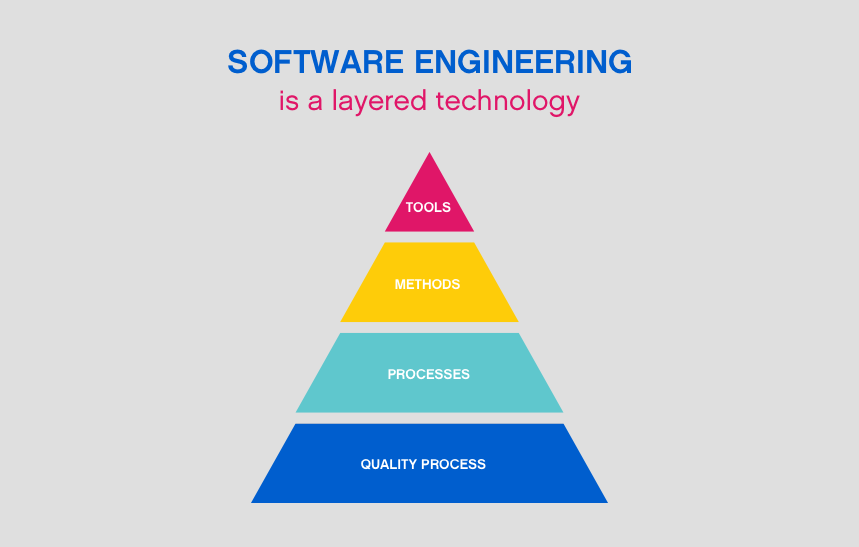
To solve this problem we are proposing an Android application that collects the data of the worker and the contractor and provides a unique ID. This application acts as an interface between worker and the contractor and help the workers and the contractor for better communication.

* This application takes the record of the all worker.
* This application also rates the worker according to their work.
* This application breaks the barrier of commuication between worker and contractor.
* A unique ID is provided to each contractor and worker so that it can be easily identified.
* This application also redues frauds with the workers. As the stipend is already declared.

**Advantages of app : -**

1. It saves time.
2. Reduce unemployment.
3. It provides workers according to need.
4. Helpful for peoples who are new in the city.

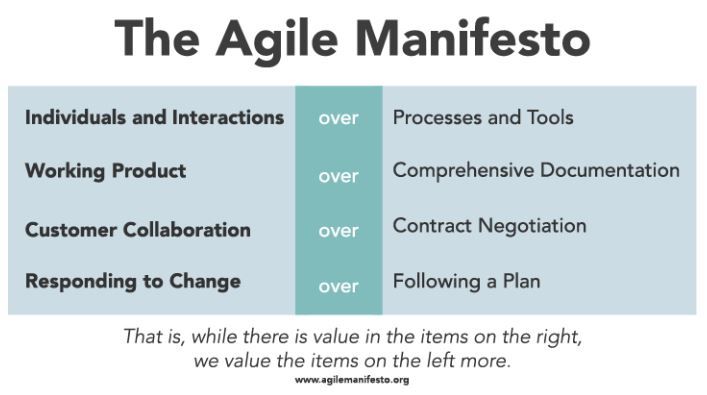
***LAYERS OF SOFTWARE ENGINERRING :-***



The above four layers of software engineering for our software is described as follows:

* TOOLS: Our software includes frontend and backend which are developed using the following technologies.
* For frontend: Android studio.
* For backend: Java and MySQL.
* METHODS: Our software allows the user to interact with each other according to their convenience. By the use of this application the worker and contractor can easily communicate with each other. This application makes easy for the contractor to hire the worker according to his/her skills and locality. And the payment of the worker is done through online medium.

**AGILE MANIFESTO**



**12 Principles behind Agile Manifesto :-**

1. Our highest priority is to satisfy the customer  
   through early and continuous delivery  
   of valuable software.
2. Welcome changing requirements, even late in  
   development. Agile processes harness change for  
   the customer's competitive advantage.
3. Deliver working software frequently, from a  
   couple of weeks to a couple of months, with a  
   preference to the shorter timescale.
4. Business people and developers must work  
   together daily throughout the project.
5. Build projects around motivated individuals.  
   Give them the environment and support they need,  
   and trust them to get the job done.
6. The most efficient and effective method of  
   conveying information to and within a development  
   team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.  
   The sponsors, developers, and users should be able  
   to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence  
   and good design enhances agility.
10. Simplicity--the art of maximizing the amount  
    of work not done--is essential.
11. The best architectures, requirements, and designs  
    emerge from self-organizing teams.
12. At regular intervals, the team reflects on how  
    to become more effective, then tunes and adjusts  
    its behaviour accordingly.

**COMPARATIVE STUDY BETWEEN VARIOUS MODELS**



**Prototype Model :-**



Prototype Model can be applied successfully to the project as the requirements are not fully clarified.

The following pointers describe the typical scenarios where Prototype Model can be used −

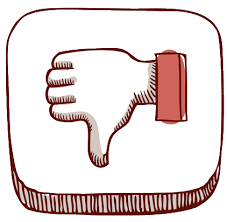
* You should use Prototyping when the requirements are unclear
* It is important to perform planned and controlled Prototyping.
* Regular meetings are vital to keep the project on time and avoid costly delays.
* The users and the designers should be aware of the prototyping issues and pitfalls.
* At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.
* In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.
* You should select the appropriate step size for each version.
* Implement important features early on so that if you run out of the time, you still have a worthwhile system

**Advantages of the Prototyping Model :-**



* Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
* Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
* Helps team member to communicate effectively
* Customer satisfaction exists because the customer can feel the product at a very early stage.
* There will be hardly any chance of software rejection.
* Quicker user feedback helps you to achieve better software development solutions.
* Allows the client to compare if the software code matches the software specification.
* It helps you to find out the missing functionality in the system.
* It also identifies the complex or difficult functions.
* Encourages innovation and flexible designing.
* It is a straightforward model, so it is easy to understand.
* No need for specialized experts to build the model

**Disadvantages of the** **Prototyping Model :-**



* Prototyping is a slow and time taking process.
* The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
* Prototyping may encourage excessive change requests.
* Sometimes customers may not be willing to participate in the iteration cycle for the longer time duration.
* There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
* Poor documentation because the requirements of the customers are changing.
* It is very difficult for software developers to accommodate all the changes demanded by the clients.

In our project case we are going to follow the Prototyping Model as it fulfils all the favourable conditions and requirements of Prototyping Model.

**Functional requirements : -**

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements are also called Functional.

Following are the functional requirements for the application : -

1. Worker login.
2. Contractor login.
3. Worker ratings.
4. Worker’s locality.

**Non-Functional requirements : -**

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

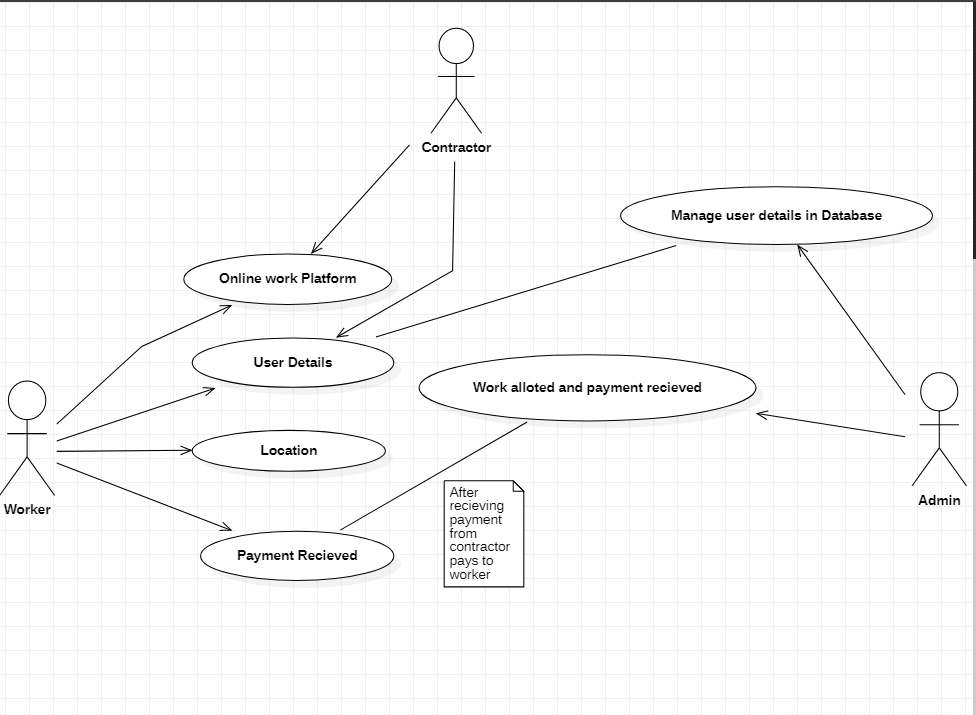
Following are the non-functional requirements for the application : -

1. A message is sent to the worker as he/she get work.
2. Users can change their login id passwords.
3. Our application is capable of handling thousands of users without affecting the performance.
4. Reliable date is provided to both ends.

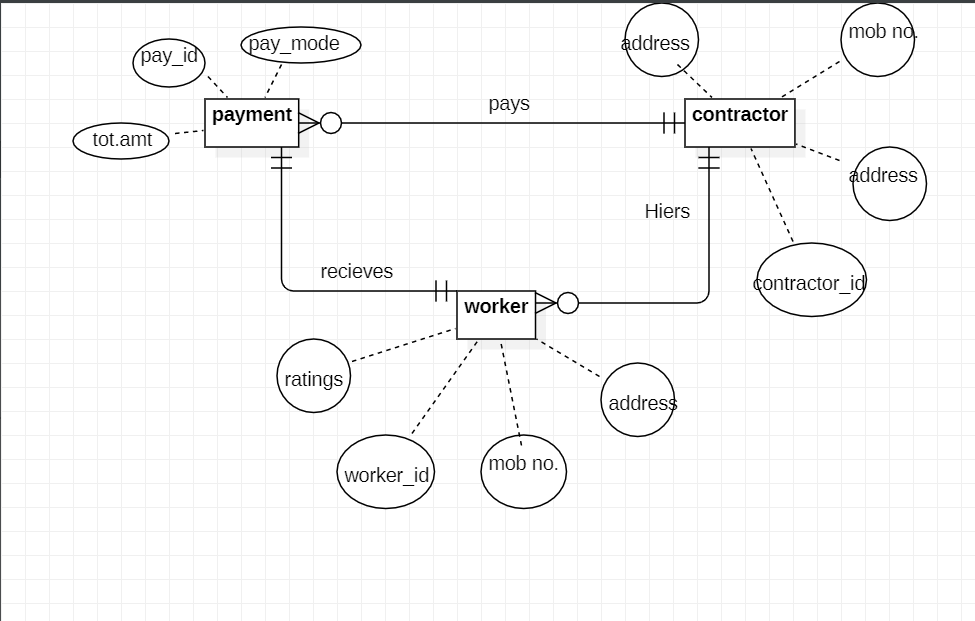
**VARIOUS SOFTWARE UML DIAGRAMS :-**

**USE CASE Diagram :-** Use Case diagrams are used to analyze the high level requirements. These requirements are expressed through different use cases. We notice three main components of this UML diagram:

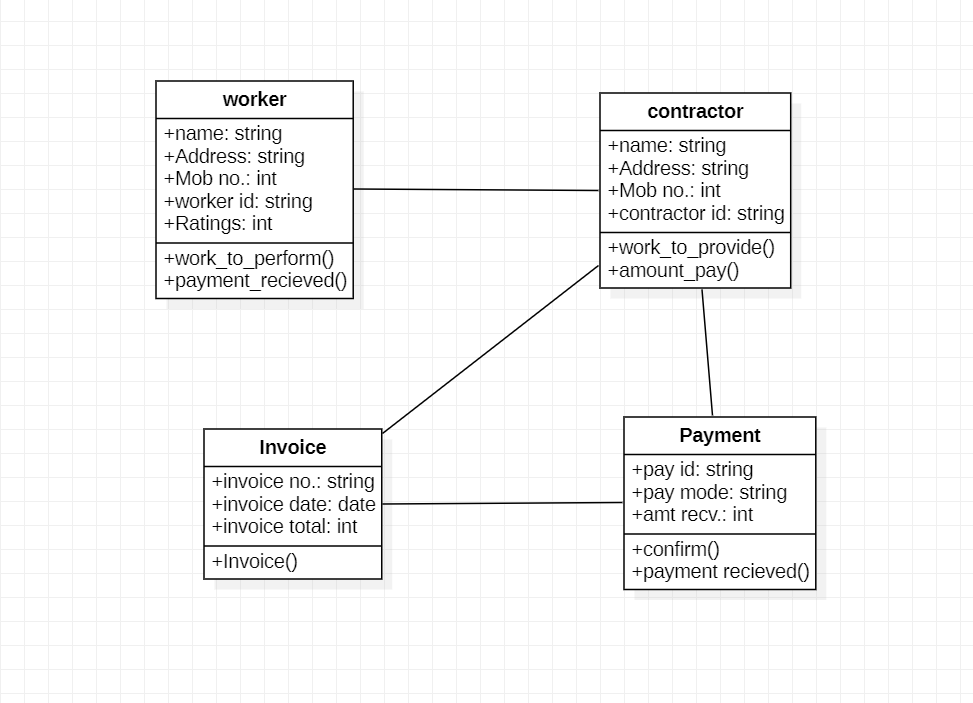
* **Functional requirements** – represented as use cases; a verb describing an action
* **Actors** – they interact with the system; an actor can be a human being, an organization or an internal or external application
* **Relationships** between actors and use cases – represented using straight arrows



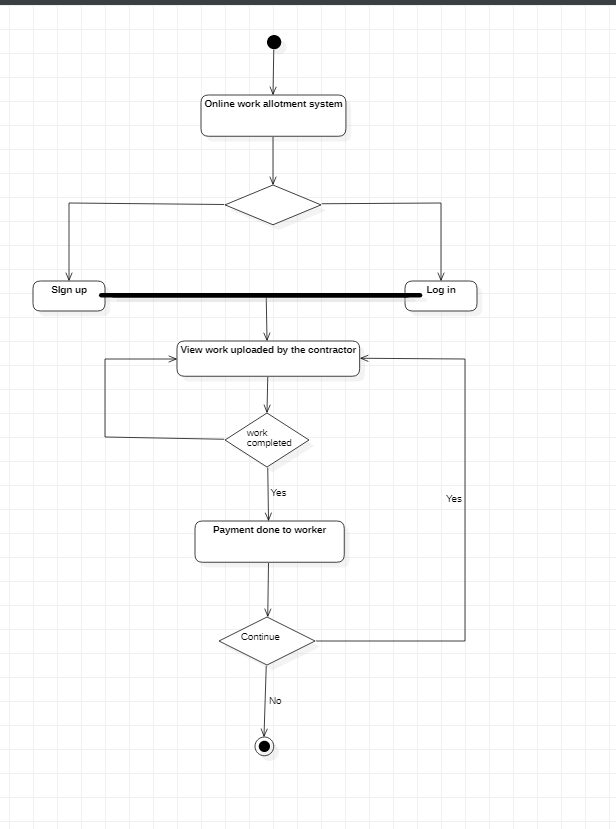
* **ER Diagram:** Entity relationship diagrams are used in software engineering during the planning stages of the software project. They help to identify different system elements and their relationships with each other. It is often used as the basis for data flow diagrams or DFD’s as they are commonly known.



* **CLASS DIAGRAM:** Class UML diagram is the most common diagram type for software documentation. Since most software being created nowadays is still based on the Objected-oriented programming paradigm, using class diagrams to document the software turns out to be a common-sense solution. This happens because OOP is based on classes and the relations between them.

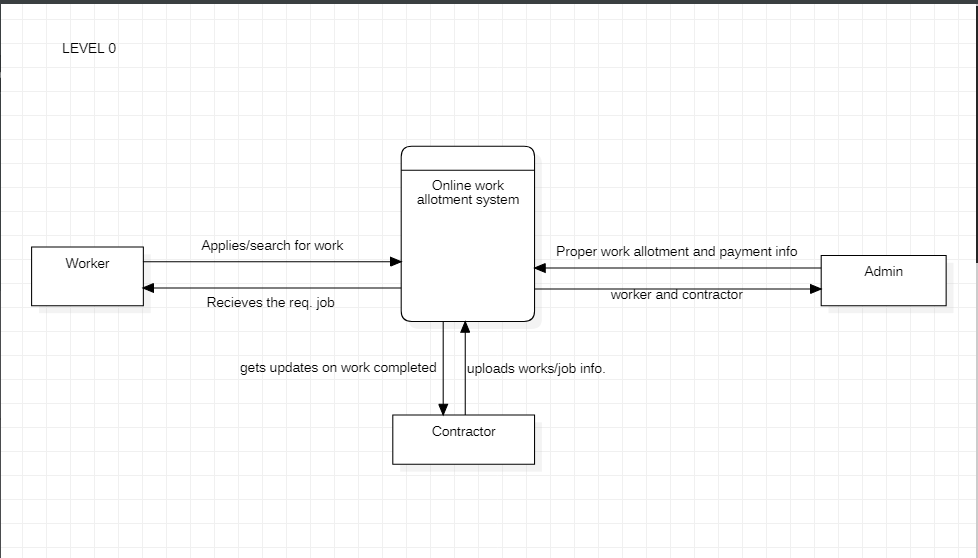


* **ACTIVITY DIAGRAM:** Activity diagrams are probably the most important UML diagrams for doing business process modelling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced by an activity and the relationship between the different activities. The entire above are essential in business process modelling.

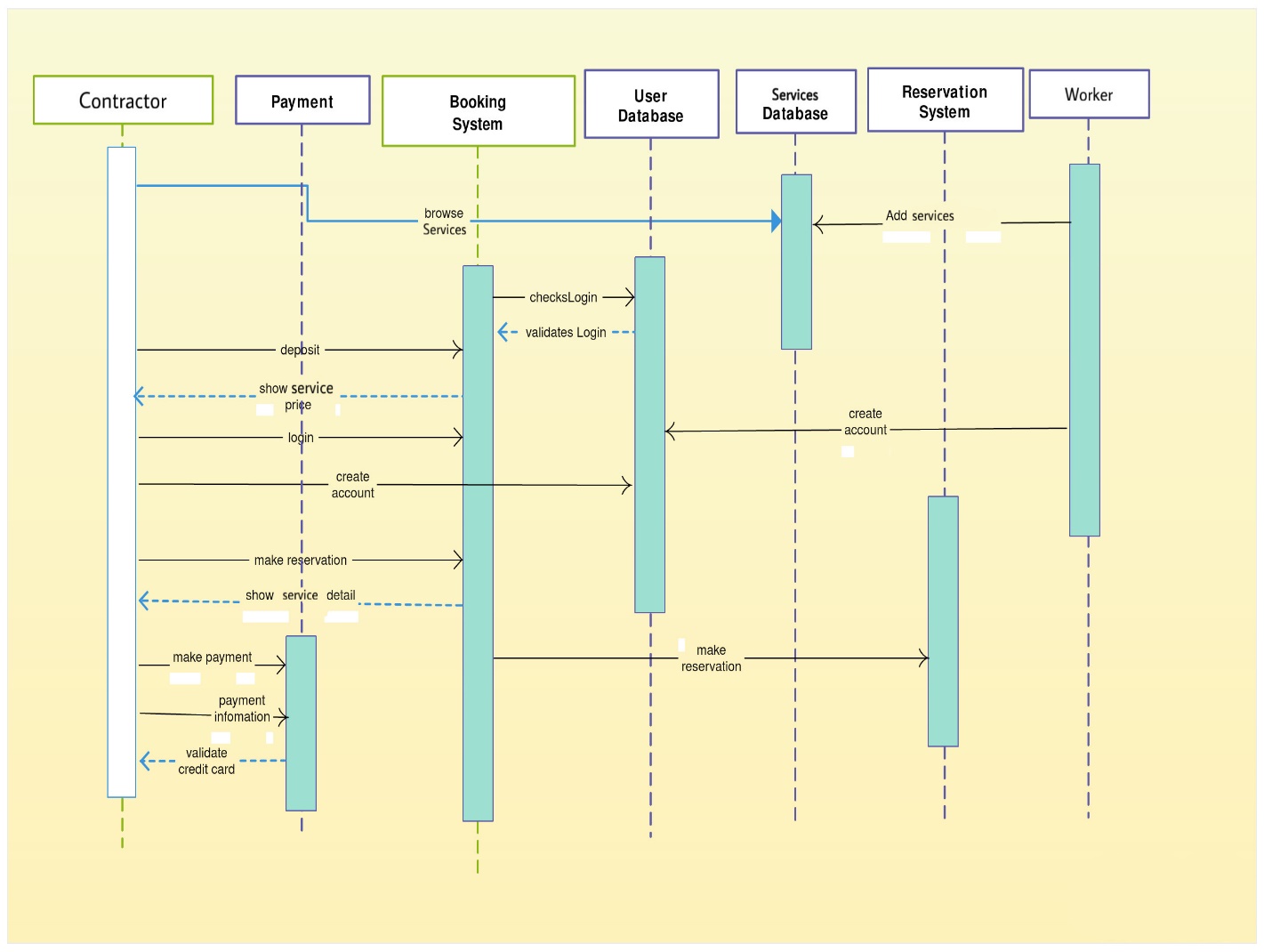


* **DATA FLOW DIAGRAM:** A data flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself.

**LEVEL 0 :-**



* **SEQUENCE DIAGRAM:** As the name suggests, sequence diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All communication is represented in a chronological manner.



**IMPLEMENTATION :-**



For building the Application and invoice generator we have a planned implementation process that will include the use of different technologies and the software building process is divided into parts as according to the prototype model the project will be delivered in parts in each iterations.

Some of the technologies and special libraries used in our project are as follows:

# **Android Studio**

# **Java Programming**

# **MySQL for database management.**



For this project the development frontend is made using Android Studio. And functionality is obtained by Java Programming. For database management MySQL is used.

By the combination of these three technologies we will develop our android application. Android studio gives the functionality to design the front end of the application. Proper design of front end is very necessary to make the application interactive with the users.

What is the behaviour of the application while clicking on a button?, what are the changes made at run time of the application?. The solution for above problems is Java programming. As Java is a platform independent language so that it can run on any environment. The behaviour of application is provided by the Java language.

MySQL, this is use for database management. For proper functionality of the application we have to store the data of the Users(worker and contractor), so, that it is easy to gather the information about the users, and also the login, password information of the users.

As the frontend and backend of the application is ready we have to connect the backend and frontend for the functionality of the application. To do this another feature of java programming language i.e Java Database Connectivity(JDBC) is used.

## WHAT IS SOFTWARE TESTING?

**Software testing** is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

**Software Testing Definition** according to **ANSI/IEEE 1059**standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

### **WHY SOFTWARE TESTING?**

## Emerging Trends in Software Testing - QC More

Some of the reasons why software testing becomes very significant and integral part in the field of information technology are as follows.

1. Cost effectiveness
2. Customer Satisfaction
3. Security
4. Product Quality

**WHEN TO STOP TESTING?**



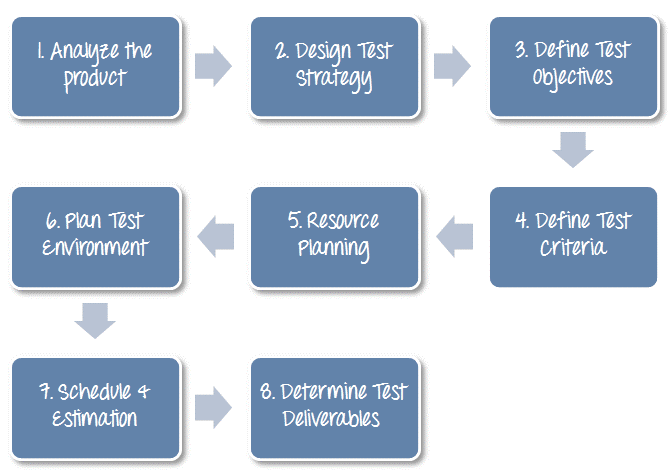
This can be difficult to determine. Most modern software [applications](http://www.360logica.net/2009/01/general-web-apolication-testing-at.html) are so complex, and run in such an interdependent environment, that complete testing can never be done. Common factors in deciding when to stop are:

* Deadlines (release deadlines, testing deadlines, etc.)
* Test cases completed with certain percentage passed
* Test budget depleted
* Coverage of code/functionality/requirements reaches a specified point
* Bug rate falls below a certain level
* Beta or alpha testing period ends

**SOFTWARE TEST PLAN..**

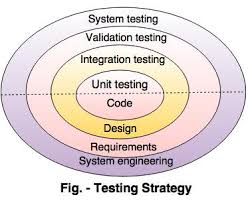
A **TEST PLAN** is a detailed document that describes the test strategy, objectives, schedule, estimation and deliverables and resources required for testing.

**HOW TO WRITE A TEST PLAN..?**



**STRATEGY OF TESTING:**

A strategy of software testing is shown in the context of spiral.  
**Following figure shows the testing strategy:**

****

* **Unit testing**  
  Unit testing starts at the centre and each unit is implemented in source code.
* **Integration testing**  
  An integration testing focuses on the construction and design of the software.
* **Validation testing**  
  Check all the requirements like functional, behavioural and performance requirement are validate against the construction software.
* **System testing**  
  System testing confirms all system elements and performance are tested entirely.

**COST ESTIMATION AND MAINTAINANCE**



**COST ESTIMATIONS:**

Cost estimation can be defined as the approximate judgments of the costs for project. Cost estimation is usually measured in terms of effort. The effort is the amount of time for one person to work for a certain period of time. COCOMO is one the most widely used software estimation models in the world. The Constructive Cost Model (COCOMO) is a procedural software cost estimation model .COCOMO is used to estimate size, effort and duration based on the cost of the software.

COCOMO predicts the effort and schedule for a software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity.

COCOMO has three different models that reflect the complexities:

**Basic Model:** This model would be applied early in a projects development. It will provide a rough estimate early on that should be refined later on with one of the other models.

**Intermediate Model:** This model would be used after you have more detailed requirements for a project.

**Detailed Model:** When design of the project is complete you can apply this model to further refine your estimate.

Within each of these models there are also three different modes. The mode you choose will depend on your work environment, and the size and constraints of the project itself.

The modes are:

**Organic:** This mode is used for “relativity small software teams developing software in a highly familiar, in-house environment”.

**Embedded:** Operating within tight constraints where the product is strongly tied to a “complex of hardware, software, regulations and operational procedures.

**Semi-detached:** An intermediate stage somewhere in between organic and embedded. Projects are usually of moderate size of up to 300,000 lines of code.

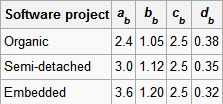
**TYPES OF COCOMO MODEL :-**

**Basic Model:** The basic COCOMO model estimates the software development effort using only Lines Of Code (LOC).

Various equations in this model are:

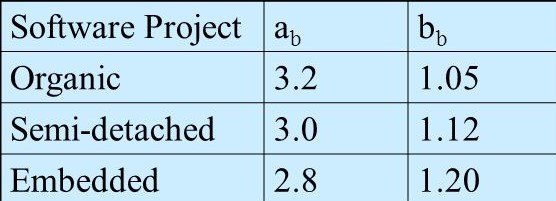
Effort Applied (E) = ab (KLOC)bb [man-months]  
 Development Time (D) = cb (Effort Applied)db[months]   
 People required (P) = Effort Applied / Development Time [count]

Where, KLOC is the estimated number of delivered lines (expressed in thousands) of code for project. The coefficients ab, bb, cb and db are given in the following table.



**Intermediate Model:** This is an extension of basic COCOMO model. This estimation model makes use of set of cost driver attributes to compute the cost of software. The formula for effort calculation is:

E= ai(KLOC) (bi)(EAF)   
Where E is the effort applied in person-months, KLOC is the estimated number of thousands of delivered lines of code for the project, and EAF is the factor calculated above. The coefficient ai and the exponent bi are given in the next table.



**Detailed Model:** Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

The detailed model uses different effort multipliers for each cost driver attribute. These Phase Sensitive effort multipliers are each to determine the amount of effort required to complete each phase. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software life cycle.

**MAINTAINANCE**



The feedback team will take regular feedback from the customer and will improve the software according to the feedback given by the customer. The maintenance team will make ensure that the software will improves continuously based on the customer feedback and also that the software works properly and give proper output. It will also responsible to fix any kind of bugs that may occur later in the software.

