# 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

**Software Engineering Project Report**



**PREPARED BY-**

**HARSHIT GUPTA**

**ISHIKA JAIN**

**KARTIK GARG**

**ONLINE BILL GENERATOR**

SECTION 1:

**PROBLEM STATEMENT**



First, waiting for invoice usually takes time and make customer waiting. Therefore, the customer's time gets wasted in waiting. The slowness of traditional bills is one of the important problems. Secondly, the cost of the paper-backed documents is another problem that faces traditional bills .It takes more effort and physical space to keep track of paper documents, to find information and to keep details secure.

* Today a user has to face many problem regarding online shopping and a proper invoice. The merchants dealing in large amount in terms of money have to face problem regarding a proper bill.
* Also customers have to carry the physical bill with them during travelling which becomes a hectic thing to manage.
* Customers cannot have a digital verifiable bill.
* It takes lots of time to generate physical bills.
* Sometimes customer has to go through Readability issues.
* There are chances to get Human error.
* It is more costly.
* It is not eco-friendly.

**SOLUTION**

****

Our software named as digital bill generator which generates the digital and E-verifiable bill and makes the shopping experience more easy which has the following Advantages:

* This project solves the customer problem of having a proper bill.
* It provides more accurate bill as, there are no chances of human error.
* It is eco-friendly in nature.
* It saves time and provides a more accurate bill in less time.
* It solves the customer problem of carrying a physical bill.
* It provides a more secure and verifiable bill.
* It makes user shopping experience more comfortable.
* It provides a long lasting bill, as customer can download it any time.
* It removes readability issues.
* It is less expensive.

**FOUR LAYERS OF SOFTWARE ENGINERRING**



**Divided into 4 layers:-**

**1. A quality Process :-**

Any engineering approach must rest on an quality. The "Bed Rock" that supports software Engineering is Quality.

**2. Process :-**

Foundation for SE is the Process Layer SE process is the glue that holds all the technology layers together and enables the timely development of computer software.

It forms the base for management control of software project.

**3. Methods :-**

SE methods provide the "Technical Questions" for building Software. Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

**4. Tools :-**

SE tools provide automated or semi-automated support for the "Process" and the "Methods". Tools are combined and interrelated so that information created by one tool can be used by another.

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

* TOOLS: Our software includes frontend and backend which are developed using the following technologies.
* For frontend: HTML, CSS, Java script.
* For backend: python 3.
* METHODS: Our software allows the customer to shop online according to their convenience and after customer successfully purchase an item the software automatically generates a digital verifiable bill and sends the copy of the bill to the customer’s E-mail address. The software also allows the customer to download the bill in future also by logging in their respective accounts. The software also generates a QR code which a customer can keep and on scanning the QR code the bill will be automatically downloaded to their system.
* PROCESS:
* QUALITY PROCESS:

**AGILE MANIFESTO**



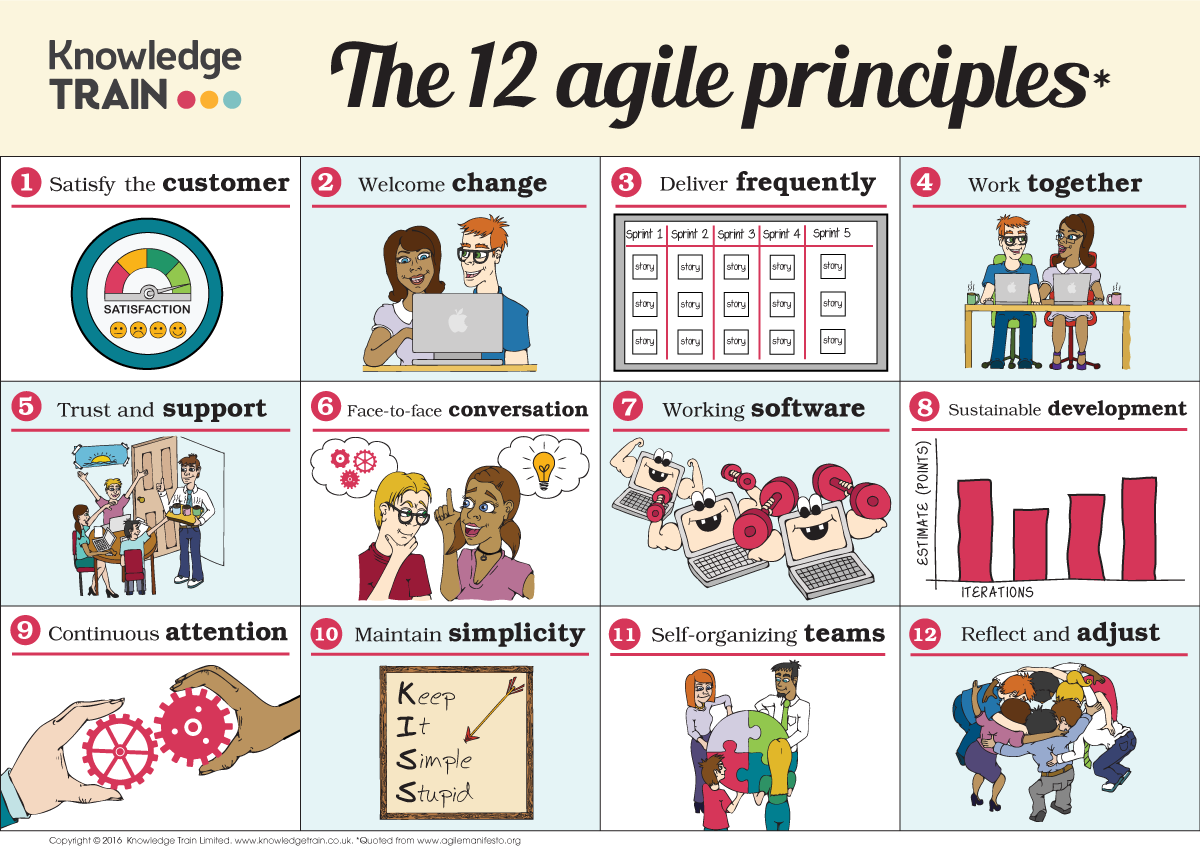
**Individuals and interactions** over processes and tools.

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation.

**Responding to change** over following a plan.

**12 Principles behind Agile Manifesto**



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

**COMPARATIVE STUDY BETWEEN VARIOUS MODELS**



**RAPID APPLICATION DEVELOPMENT MODEL (RAD)**

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail.

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

* RAD should be used only when a system can be modularized to be delivered in an incremental manner.
* It should be used if there is a high availability of designers for modeling.
* It should be used only if the budget permits use of automated code generating tools.
* RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
* Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

**PROS AND CONS of RAD Model**

RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

**ADVANTAGES OF RAD MODEL**

* Changing requirements can be accommodated.
* Progress can be measured.
* Iteration time can be short with use of powerful RAD tools.
* Productivity with fewer people in a short time.
* Reduced development time.
* Increases reusability of components.
* Quick initial reviews occur.
* Encourages customer feedback.
* Integration from very beginning solves a lot of integration issues.

**DISADVANTAGES OF RAD MODEL**

* Dependency on technically strong team members for identifying business requirements.
* Only system that can be modularized can be built using RAD.
* Requires highly skilled developers/designers.
* High dependency on modelling skills.
* Inapplicable to cheaper projects as cost of modelling and automated code generation is very high.
* Management complexity is more.
* Suitable for systems that are component based and scalable.
* Requires user involvement throughout the life cycle.
* Suitable for project requiring shorter development times.

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



**WHAT IS FUNCTIONAL REQUIREMENT?**

In software engineering, a functional requirement defines a system or its component. It describes the functions software must perform. A function is nothing but inputs, its behaviour, 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 software requirements help you to capture the intended behaviour of the system. This behaviour may be expressed as functions, services or tasks or which system is required to perform.

**Functional Requirements for Online Bill Generation and shopping system.**

* The system should provide appropriate viewers for user to read the list of items in item store.
* The system should provide user’s unique reference (USER\_ID) where customer can have personal account and make payment.
* The user shall choose what item that they want to receive.
* The system should print, for the owner a summary and show the report for who received the item and invoices.
* The system should automatically compute the bills and send the invoices with their items.
* The system should be able to manage some simple geographic info and provide company’s description.
* The system should automatically generate invoices and send a mail to the users with the invoice attached and also generate a QR code which enables the user to verify the bills online.
* The user should choose where they want the item to be delivered.

**WHAT IS NON FUNCTIONAL REQUIREMENT?**

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 is > 10000. Description of non-functional requirements is just as critical as a functional requirement.

**Non-Functional Requirements for Online Bill Generation and shopping system.**

* **Product Requirement.**

1. The user interface shall be implemented as simpleHTML, CSS, JAVASCRIPT.
2. The list of items should be listed in the order form.
3. The product should be available and updated each day.
4. Proper product details should be present.

* **Owner Requirement.**

1. The owner should provide other services on request by the owner.
2. The owner should communicate with other shop owner or other individual for the services that in the list but not provided.

* **Organizational Requirement.**

1. The system development process and deliverable reports shall conform to the process.
2. The delivery should be made every day.
3. Each item delivered must be reported and recorded each time.
4. The system should automatically generate correct bills.

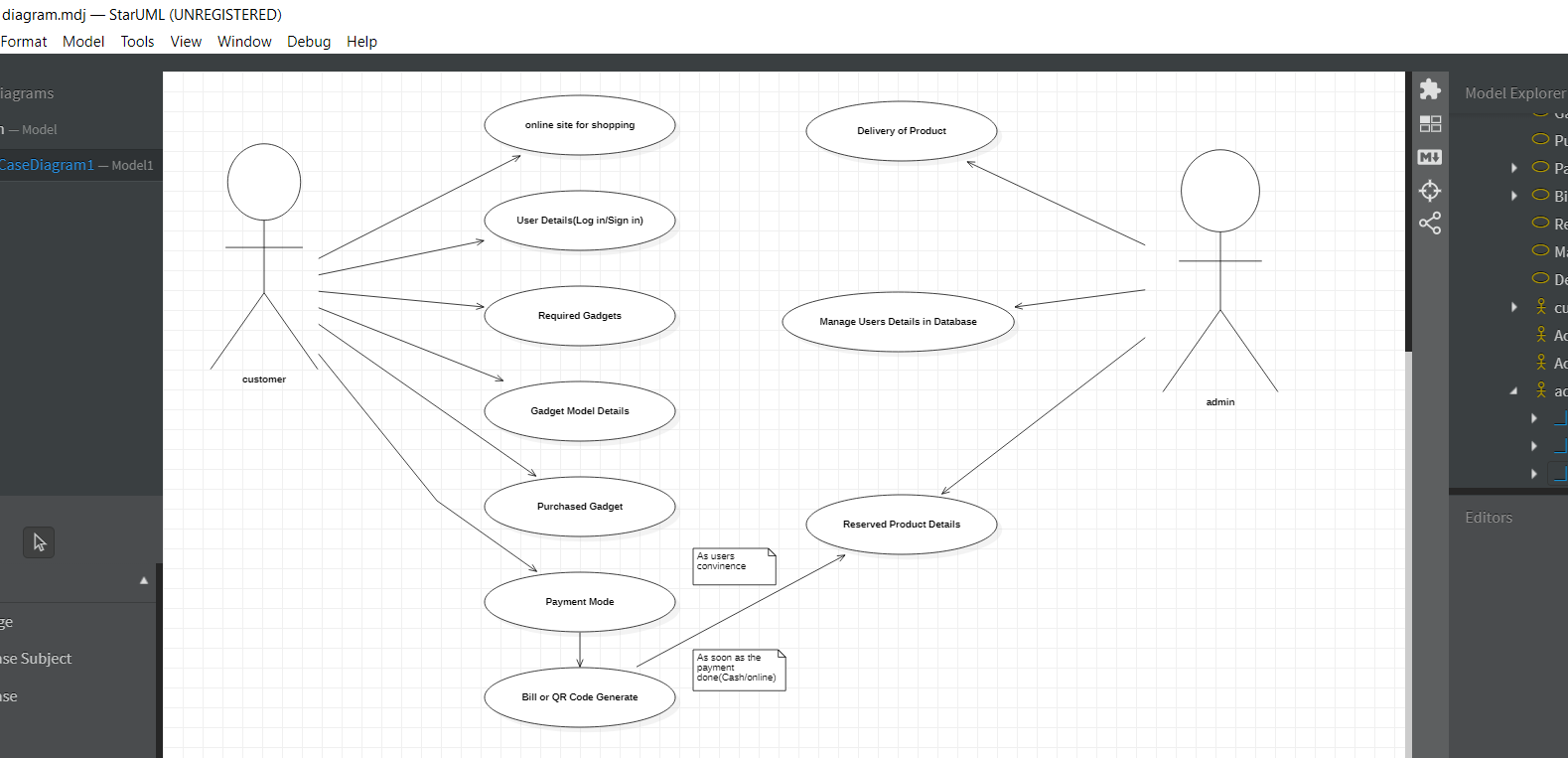
* **External Requirement.**

1. The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.
2. The system should provide geographical attributes and some information about the services.
3. The system must provide safe and secure payment environments

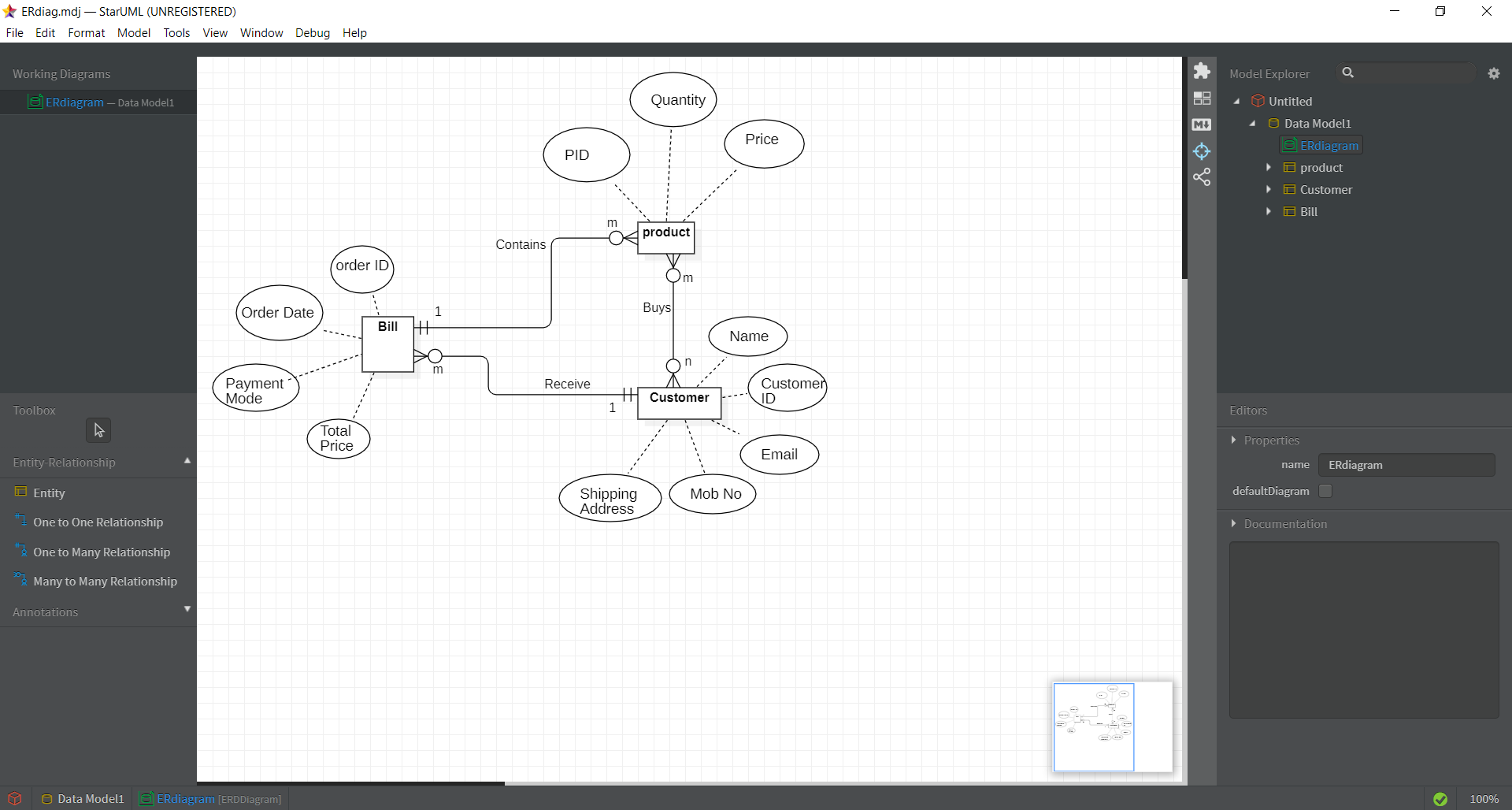
**VARIOUS SOFTWARE UML DIAGRAMS**

**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:

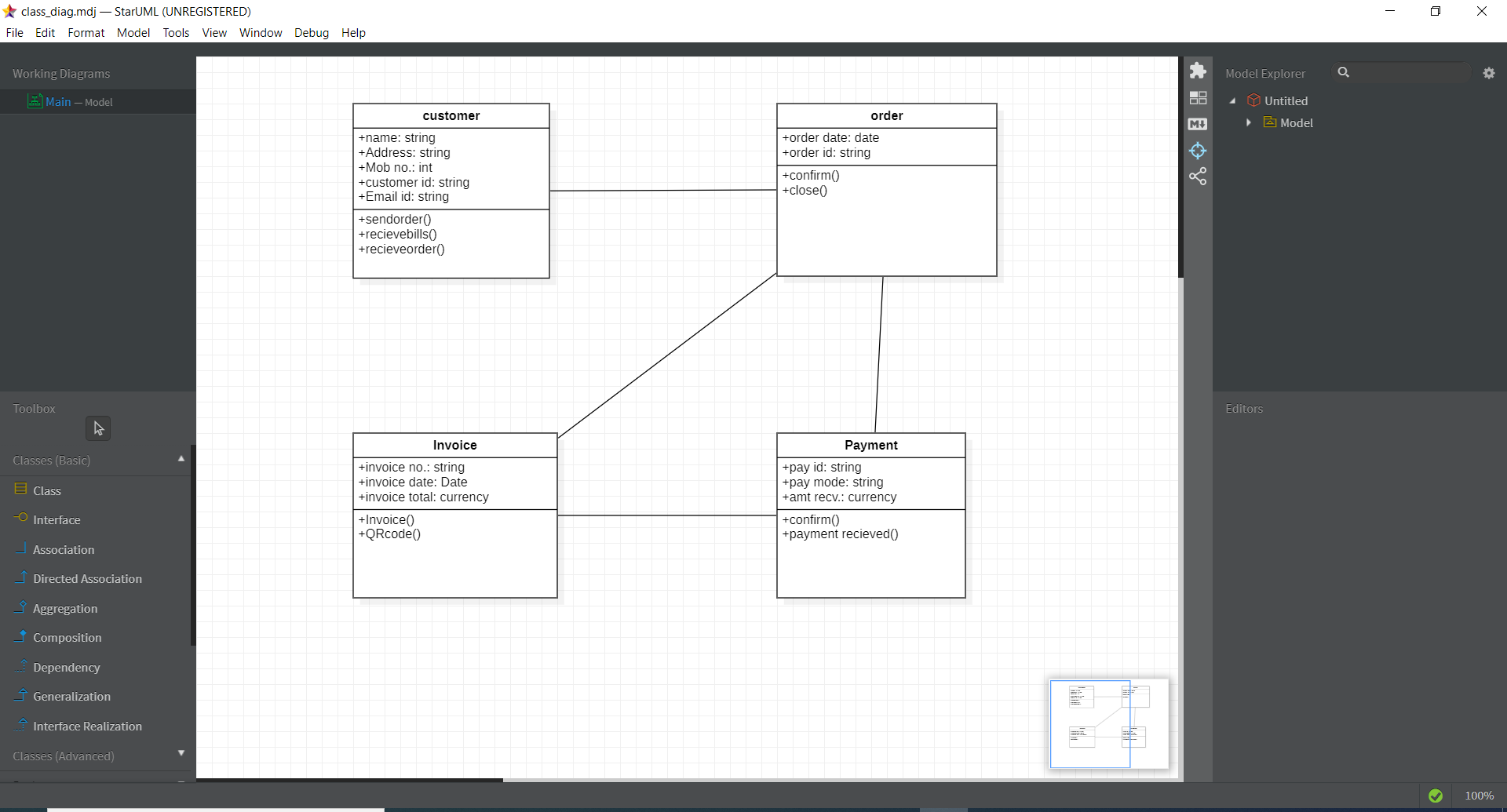
* **Functional requirements** – represented as use cases; a verb describing an action
* **Actors** – they interact with the system; an actor can be a human being, an organization or an internal or external application
* **Relationships** between actors and use cases – represented using straight arrows



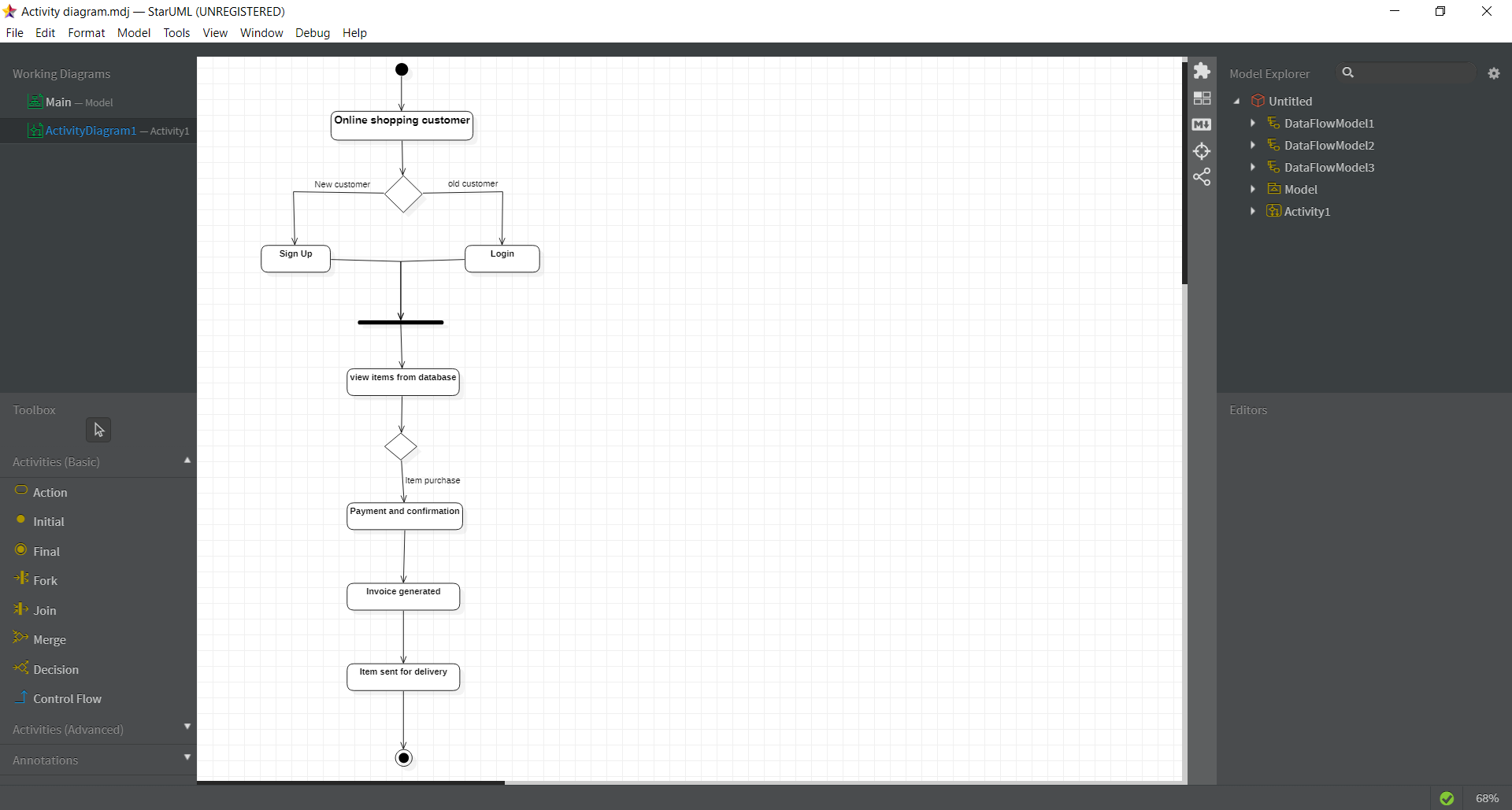
* **ER Diagram:** Entity relationship diagrams are used in software engineering during the planning stages of the software project. They help to identify different system elements and their relationships with each other. It is often used as the basis for data flow diagrams or DFD’s as they are commonly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

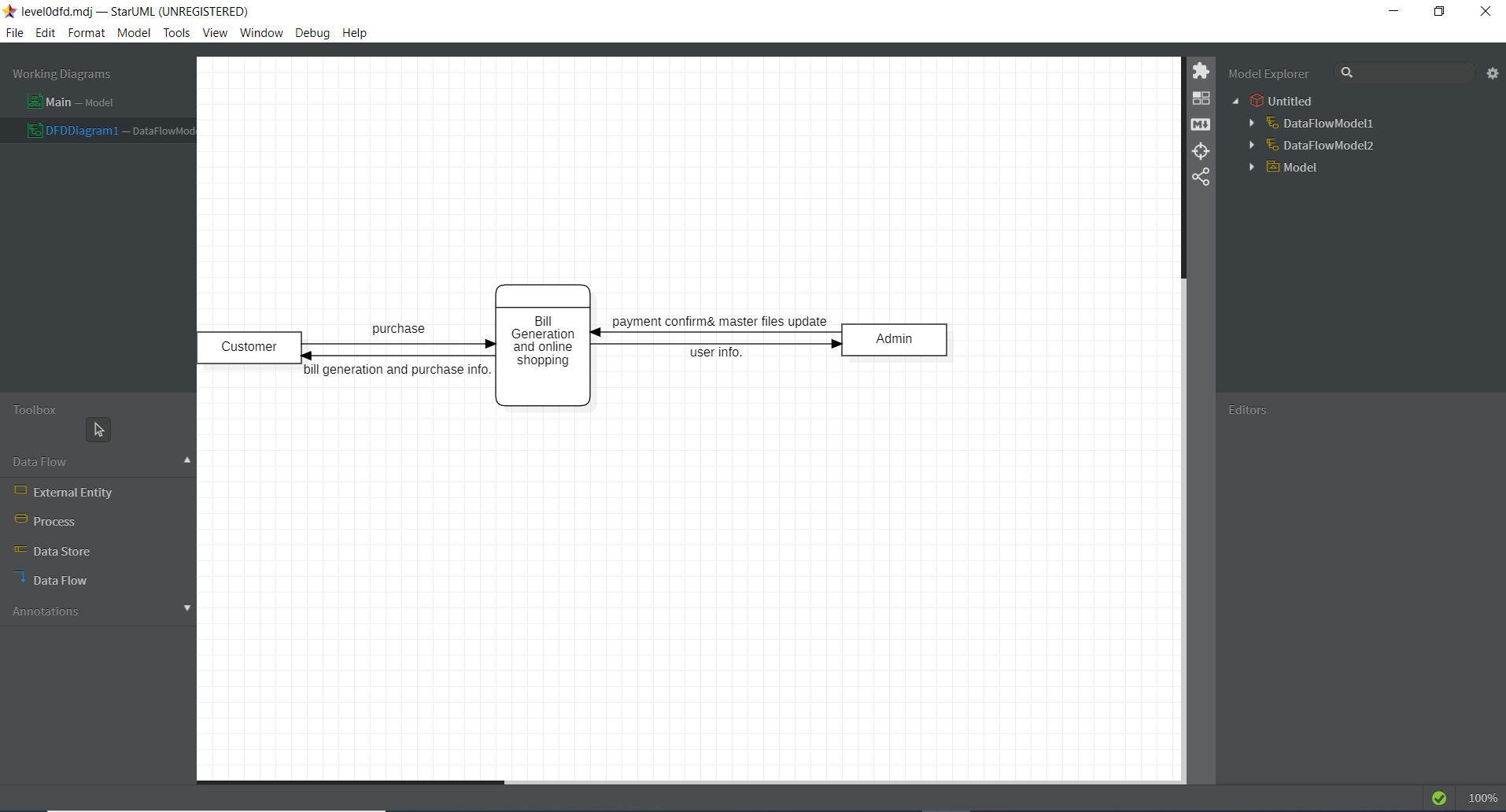
****

* **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.

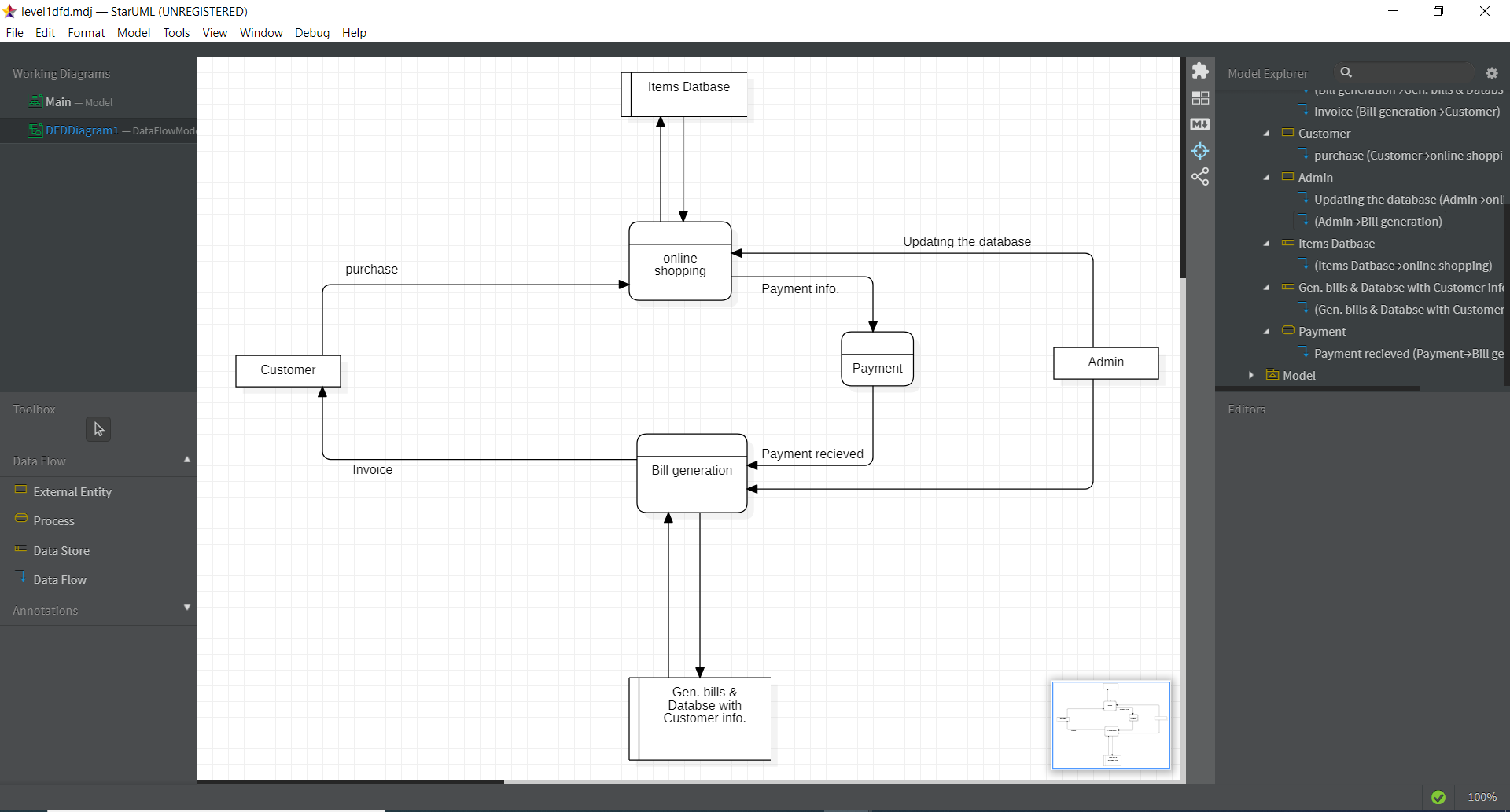


* **DATA FLOW DIAGRAM:** A data flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself.

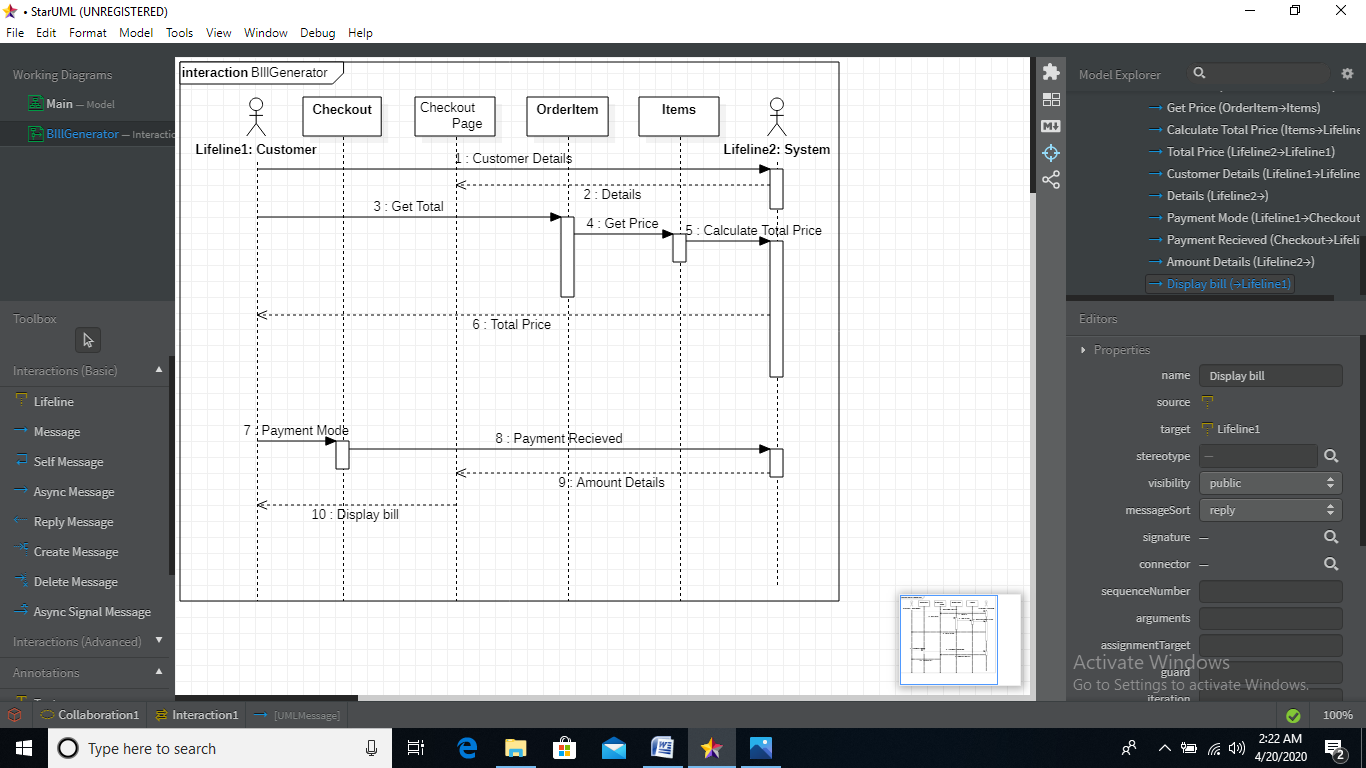
**LEVEL 0:**



**LEVEL 1:**



* **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:**



SOFTWARE IMPLEMENTATION

For building the online shopping 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 RAD 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:

* Python 3
* Pyqrcode (for the generation of qrcode.)
* Numpy library.
* Math module
* Validation using python.
* Django used as a server side technology.
* HTML 5, CSS, JavaScript
* MySQL for database management.

In this project the development part consist of frontend which is made using HTML5, CSS and JavaScript is used for validation and other purposes.

Python is used as the major technology for the development purpose and Django framework is used to make the website more responsive and user friendly. Where Data bases is handled using MySQL.

Major python libraries used for the development are Numpy for data representation and proper management. And QRCode module from pyqrcode for dealing with the QRCode related work.

In this project two major sections that are made:   
first a proper invoice generator that generates the bills that are more reliable than other handmade invoices as soon as after the successful payment.

The other part is to make friendly interface where a customer can search things online and thus according to his/her choice complete the shopping of the products.

The user information such as contact details and login information must be handled securely and appropriately and for that proper system is made here.

## 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?****

## High Quality Question Mark Wallpapers

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?**

[](http://1.bp.blogspot.com/-2FxzFV-fKr8/UzAO6fUY-TI/AAAAAAAAAZQ/EcyW1h1M5D0/s1600/when+should+stop+testing.jpg)

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:

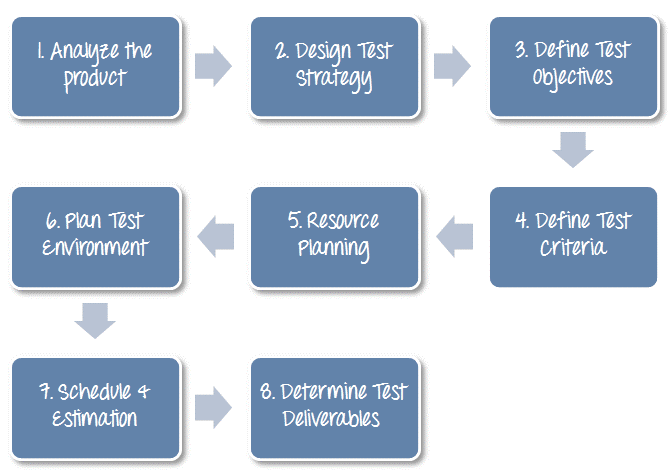
* Deadlines (release deadlines, testing deadlines, etc.)
* Test cases completed with certain percentage passed
* Test budget depleted
* Coverage of code/functionality/requirements reaches a specified point
* Bug rate falls below a certain level
* Beta or alpha testing period ends

**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.

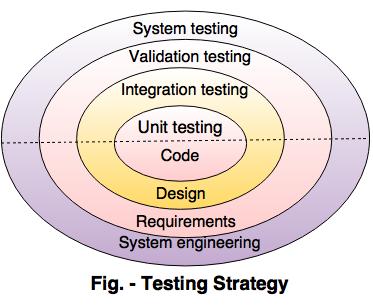
**HOW TO WRITE A TEST PLAN..?**

Follow the seven steps below to create 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.

**Testing types for bill generator:**



* **Functional Testing**: It is the act of testing the base functionality of the billing. It is to verify whether the application behaves in the same way as it is supposed to be like handling orders, calculation, an addition of VAT as per the country etc.
* **Integration**: Test integration with your credit card service.
* **Performance**: Identify various performance metrics like the highest possible number of users coming through gateways during a specific day and converting them to concurrent users.
* **Security**: You need to perform a deep security pass for QR code generator and to secure the details of customer.

**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.

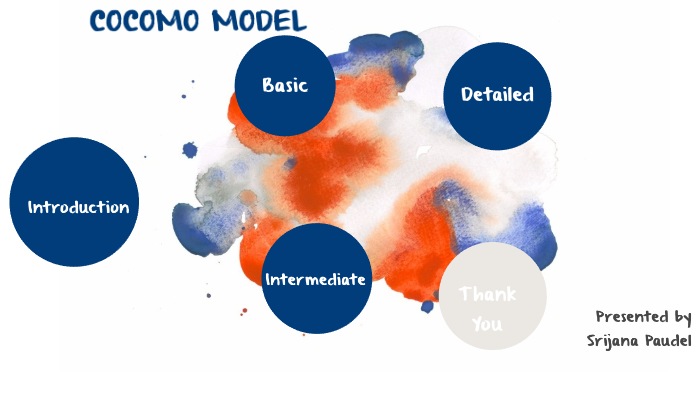
The modes are:

**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**



COCOMO

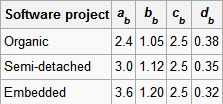
MODEL

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

Various equations in this model are:

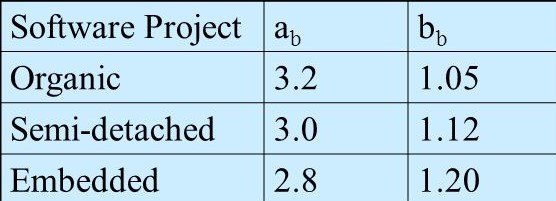
Effort Applied (E) = ab (KLOC)bb [man-months]  
 Development Time (D) = cb (Effort Applied)db[months]   
 People required (P) = Effort Applied / Development Time [count]

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.

**ESTIMATE COST OF OUR PROJECT-ONLINE BILL GENERATOR**

We are using basic COCOMO Model for our project.

Effort Applied (E) = ab (KLOC)bb [man-months]

Our software comes under the category of semi-detached project,

Where ab=3.0

bb=1.12

KLOC=2000 lines of code

From the above formula,

Effort Applied(E)=14937.39[man-month]

Development Time (D) = cb (Effort Applied)db[months]

Where cb= 2.5

db=0.35

From formula,

Development Time (D) =72.26[month]

People required (P) = Effort Applied / Development Time [count]

People required (P) =207

*Basic equation is given by:*

M = P + ke^(c-d)

Where M= total effort expended

P= productive effort that involves analysis, design coding etc..

K= constant

C= complexity measure due to lack of good design and documentation

D= degree to which maintenance team is familiar with software

In our software (Online Bill Generator)

We are assuming it to be at professional level

People Required = 207 Persons/month

Empirically determined constant =0.3

Complexity of code(design, implementation, coding ) =8

Maintenance team has good understanding about software = 0.9

Then,

P=207

K=0.3

C=8

We know that

M=P+ke^(c-d)

= 207+0.3e^(8-0.9)

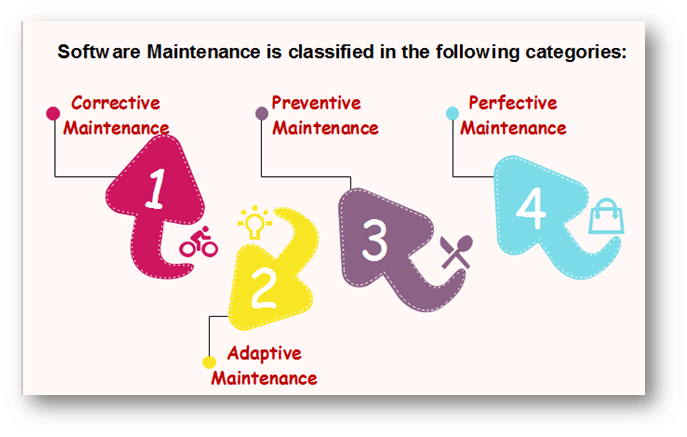
207 + 363.59 = 570.59PM

**MAINTAINANCE**



Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.

## TYPES OF SOFTWARE MAINTENANCE



### 1. Corrective Maintenance

Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

### 2. Adaptive Maintenance

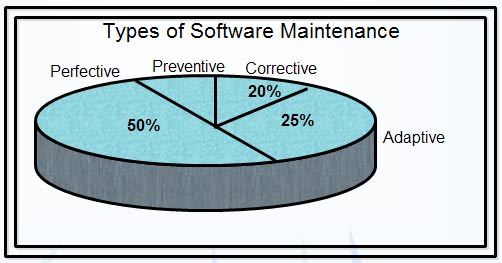
It contains modifying the software to match changes in the ever-changing environment.

### 3. Preventive Maintenance

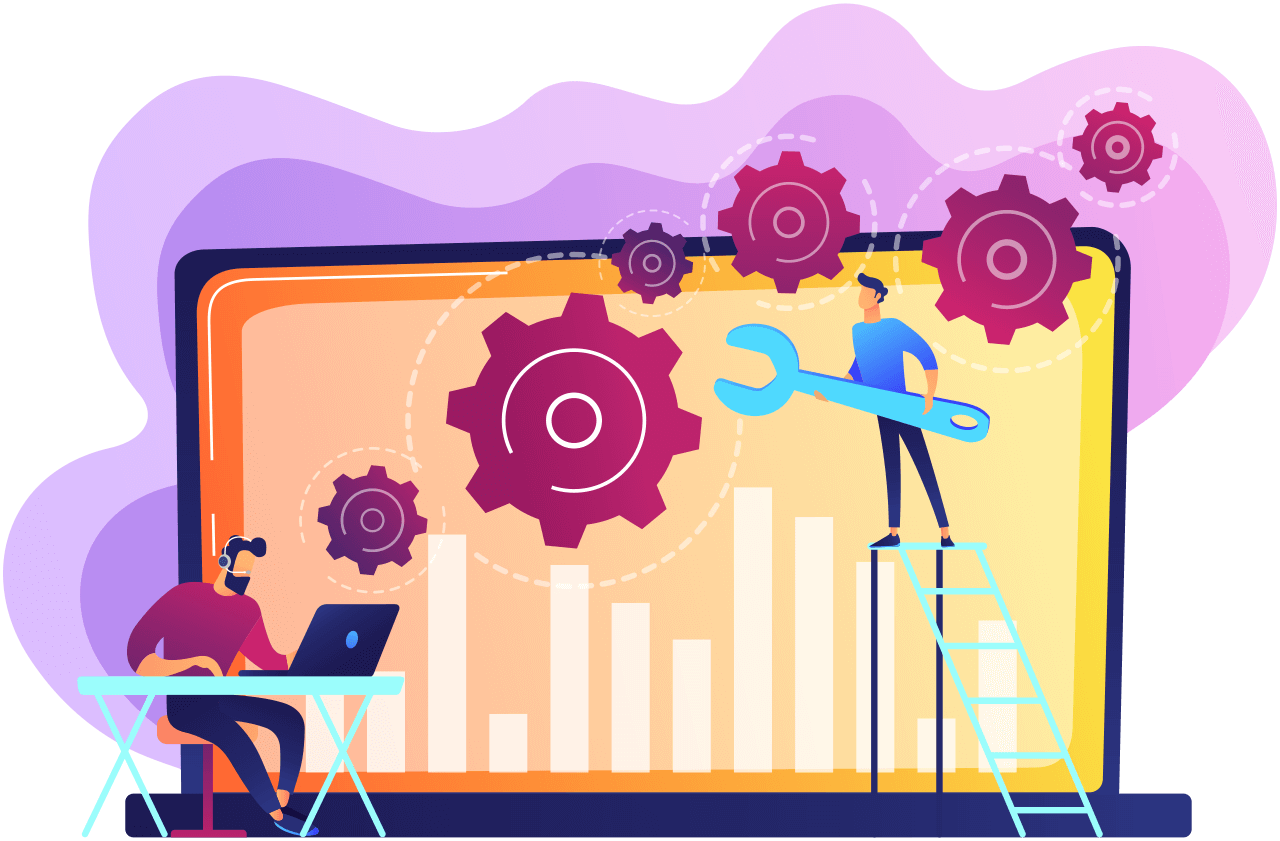
It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

### 4. Perfective Maintenance

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.



**NEED FOR MAINTENANCE–**



Software Maintenance must be performed in order to:

* Correct faults.
* Improve the design.
* Implement enhancements.
* Interface with other systems.
* Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
* Migrate legacy software.
* Retire software.

**ROLE OF MAINTENANCE IN OUR PROJECT**



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.