WHAT IS SOFTWARE(SW)

is a **set of instructions** (computer program) and its **associated documentations** that tells a computer **what to do** or **how to perform a task.**

ATTRIBUTES OF GOOD SOFTWARE

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Industry Standard SW

An industry standard SW must be:

Operational

- Budget
- Efficiency
- Usability
- Dependability
- Correctness
- Functionality
- Security

Transitional

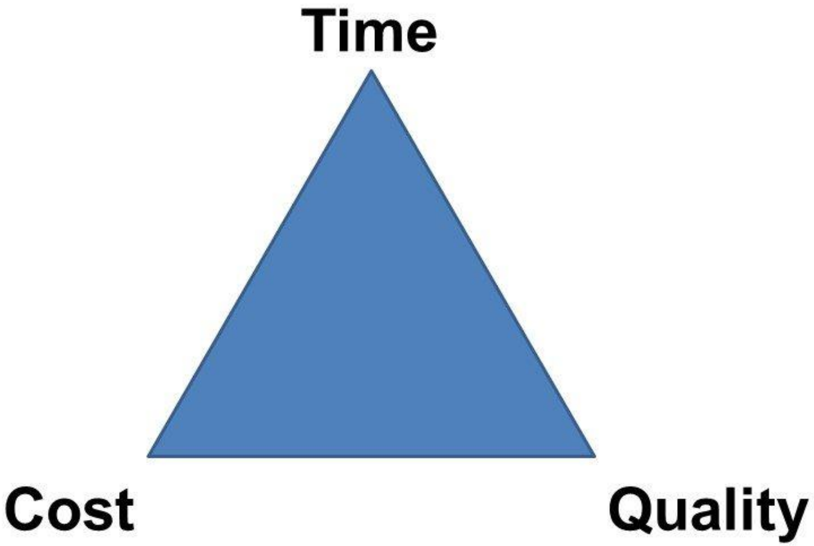
- Interoperability
- Reusability
- Portability
- Adaptability

Easy to Maintain

- Flexibility
- Maintainability
- Modularity
- Scalability

INDUSTRY STANDARD SW

- ✓ quality, cost, and schedule are the main forces that drive a industry standard software project.
- ✓ should be produced at reasonable cost, in a reasonable time, and should be of good quality.



WHAT IS SW ENGINEERING

Software engineering is defined as a discipline whose aim is the production of **quality software**, **delivered on time** and **within the budget** and that satisfies its **requirement.**

IEEE, in its standard 610.12-1990, defines software engineering as the application of a systematic, disciplined, which is a computable approach for the **development**, **operation**, and **maintenance** of software.

Key Difference

- ✓ Software Engineer applies the principles of software engineering for designing, development, maintenance, testing, and evaluation of computer software whereas Software Developer builds software which runs across various types of computer.

- ✓ Software Engineer works with other components of the hardware system whereas Software Developers write a complete program.

- ✓ Software Engineer creates the tools to develop software while Software Developers use readymade tools to build apps.

- ✓ Software Engineer tends to solve issues on a much larger scale whereas Software Developers tend to do everything that engineers do but on a limited scale.

Challenges of Software Engineering

- ✓ In safety-critical areas such as space, aviation, nuclear power plants, etc. the cost of software failure can be massive because lives are at risk.

- ✓ Increased market demands for fast turnaround time.

- ✓ Dealing with the increased complexity of software and need for new applications.

SOFTWARE ENGINEERING ETHICS

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices.

The Ethical Principles are:

1. **PUBLIC** - Software engineers shall act **consistently** with the **public interest**.
2. **CLIENT AND EMPLOYER** - Software engineers shall act in a manner that is in the **best interests of their client and employer** consistent with the public interest.
3. **PRODUCT** - Software engineers shall ensure that their products and related modifications meet the **highest professional standards** possible.
4. **JUDGMENT** - Software engineers shall maintain **integrity and independence** in their professional judgment.
5. **MANAGEMENT** - Software engineering managers and leaders shall subscribe to and **promote an ethical approach** to the management of software development and maintenance.
6. **PROFESSION** - Software engineers shall advance the **integrity and reputation** of the profession consistent with the public interest.
7. **COLLEAGUES** - Software engineers shall be **fair to and supportive** of their colleagues.
8. **SELF -** Software engineers shall participate in **lifelong learning** regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

https://ethics.acm.org/code-of-ethics/software-engineering-code/

SW Process

- ✓ software process is a set of activities that leads to the production of a software product.
- ✓ There is no ideal process, and many organizations have developed their own approach to software development.
- ✓ For some systems, such as critical systems, a very structured development process is required. For business systems, with rapidly changing requirements, a flexible, agile process is likely to be more effective.
- ✓ fundamental activities that are com-mon to all software processes:

  1. Software specification The functionality of the software and constraints on its operation must be defined.
  2. Software design and implementation The software to meet the specification must be produced.
  3. Software validation The software must be validated to ensure that it does what the customer wants.
  4. Software evolution - The software must evolve to meet changing customer needs.

SW Size Estimation

- ✓ Software size is an important metric to be used for various purposes during SE dev projects
- ✓ Difficult to Measure
- ✓ Various approaches are used to measure SW size
    - **LINE OF CODE (LOC)**
    - **TOKEN COUNT (HALSTEAD PRODUCT METRICS)**
    - **FUNCTION POINT ANALYSIS (FPA)**

SW Size Estimation

LINE OF CODE (LOC)

- ✓ This metric is based on the number of lines of code present in the program.
- ✓ The lines of code are counted to measure the size of a program.
- ✓ The comments and blank lines are ignored during this measurement.
- ✓ The LOC metric is often presented on thousands of lines of code (KLOC).

SW Size Estimation

LINE OF CODE (LOC)

- ✓ This metric is based on the number of lines of code present in the program.
- ✓ The lines of code are counted to measure the size of a program.
- ✓ The comments and blank lines are ignored during this measurement.
- ✓ The LOC metric is often presented on thousands of lines of code (KLOC).

SW Size Estimation

TOKEN COUNT (HALSTEAD PRODUCT METRICS)

- ✓ LOC is not consistent, because all lines of code are not at the same level..
- ✓ Some lines are more difficult to code than others.
- ✓ **Halstead** stated that any software program could be measured by counting the number of **operators** and **operands**.
- ✓ From these set of operators and operands, he defined a number of formulae to calculate the **vocabulary**, the **length**, and the **volume** of the software program.

SW Size Estimation

TOKEN COUNT (HALSTEAD PRODUCT METRICS)

✓ **Halstead Metric**

Program Vocabulary

✓ It is the number of **unique** operators plus the number of unique operands as given below:

$n = n1 + n2$

where n = program vocabulary

n1 = number of unique operators

n2 = number of unique operands

SW Size Estimation

TOKEN COUNT (HALSTEAD PRODUCT METRICS)

✓ **Halstead Metric**

Program Length

✓ It is the **total usage of all** the operators and operands appearing in the implementation.

N = N1 + N 2

where N = program length

N1 = all operators appearing in the implementation

N2 = all operands appearing in the implementation

SW Size Estimation

TOKEN COUNT (HALSTEAD PRODUCT METRICS)

✓ **Few Halstead Metric**

Program Volume

✓ The volume refers to the size of the program and it is defined as the program length times the logarithmic base 2 of the program vocabulary.

$V = N \log_2 n$

where V = program volume

N = program length

n = program vocabulary

SW Size Estimation

FUNCTION POINT ANALYSIS (FPA)

- ✓ It is based on the idea that the software size should be measured according to the functionalities specified by the user.
- ✓ Therefore, FPA is a standardized methodology for measuring various functions of a software from the user’s point of view.
- ✓ The size of an application is measured in **function points**.

SW Size Estimation

FUNCTION POINT ANALYSIS (FPA)

Process used to calculate the function points

1. Determine the type of project for which the function point count is to be calculated. For example, development project (a new project) or enhancement project.

2. Identify the counting scope and the **application boundary.**

3. Identify **data functions** (internal logical file and external interface files) and their **complexity.**

4. Identify **transactional functions** (external inputs, external outputs, and external queries) and their **complexity.**

5. Determine the **unadjusted function point count (UFP).**

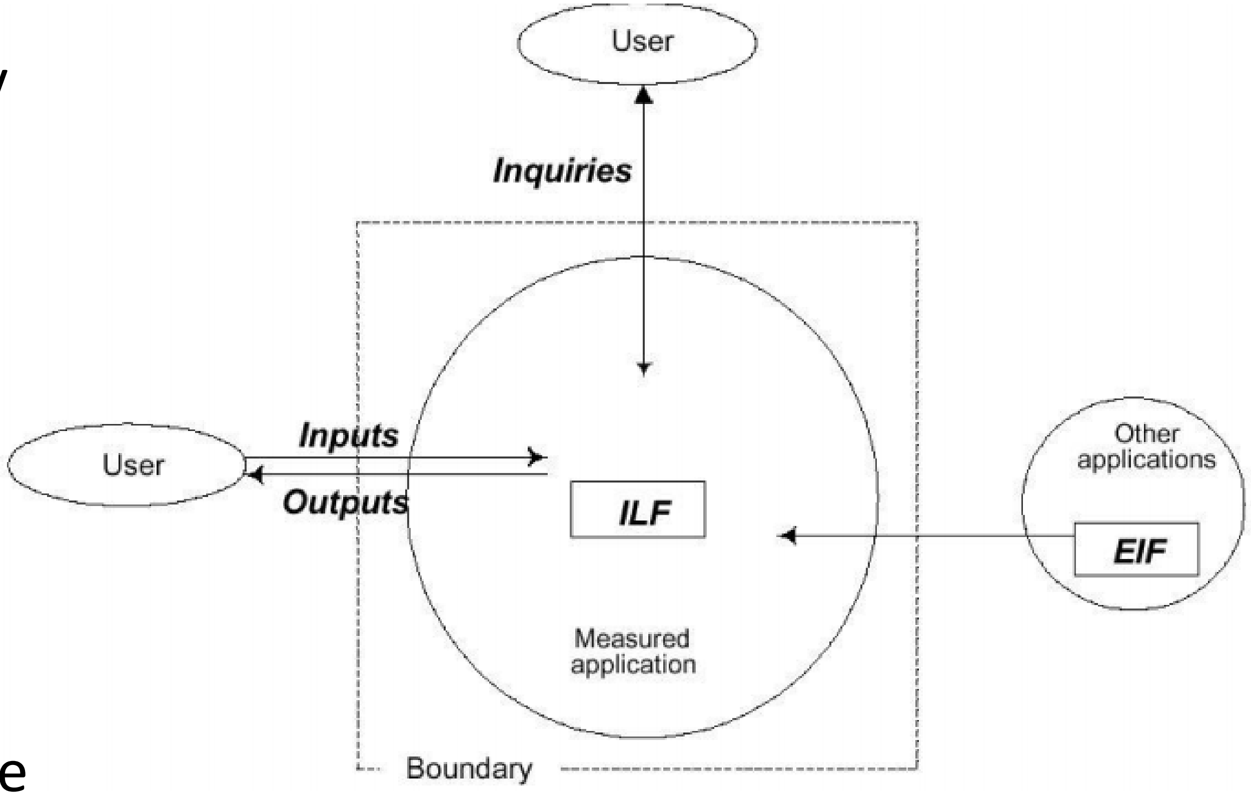
6. Determine the **value adjustment factor (VAF)**, which is based on **14 general system characteristics (GSCs).**

7. Calculate the **adjusted function point count (AFP).**

SW Size Estimation

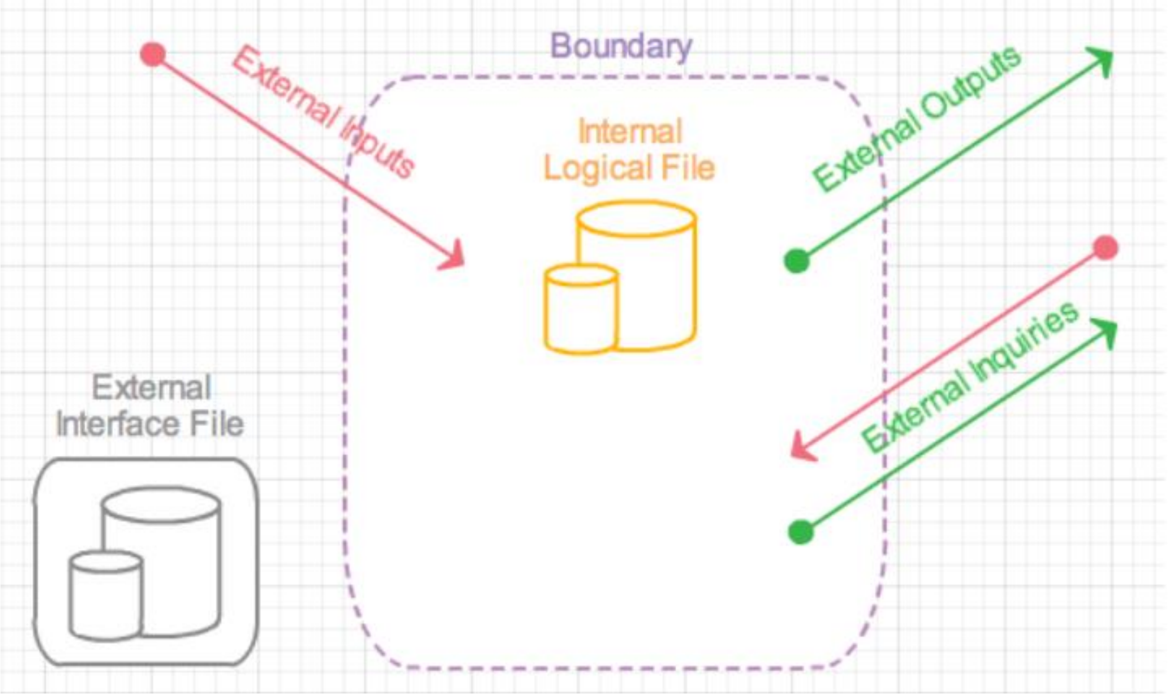
FUNCTION POINT ANALYSIS (FPA)

- ✓ The first step in calculating FP is to identify the **counting boundary**.
- ✓ Counting boundary: The border between the application or project being measured and external applications or the user domain.
- ✓ A boundary establishes which functions are included in the function point count



SW Size Estimation

Size Metric: FUNCTION POINT ANALYSIS (FPA)



✓ Two data function types:

- **Internal Logical Files (ILF):** This is the set of data present within the system. The majority of the data will be interrelated and are captured via the inputs received from the external sources.
- **External Interface Files(EIF):** represent the data that your application will use/reference, but data that is not maintained by your application.

✓ Three Transactional function types:

- **External Inputs (EI)**: these are end-user actions such as putting in a login or executing a mouse click.
- **External Outputs (EO):** the system provides the end-user output or interface such as a GUI display or items in a report.
- **External Inquiries (EQ):** this function is initiated by the end-user. For example, the end-user wishes to submit a query to a database or requests on-line help. In any case the developer provides a means for the end-user to "search" for answers.

SW Size Estimation

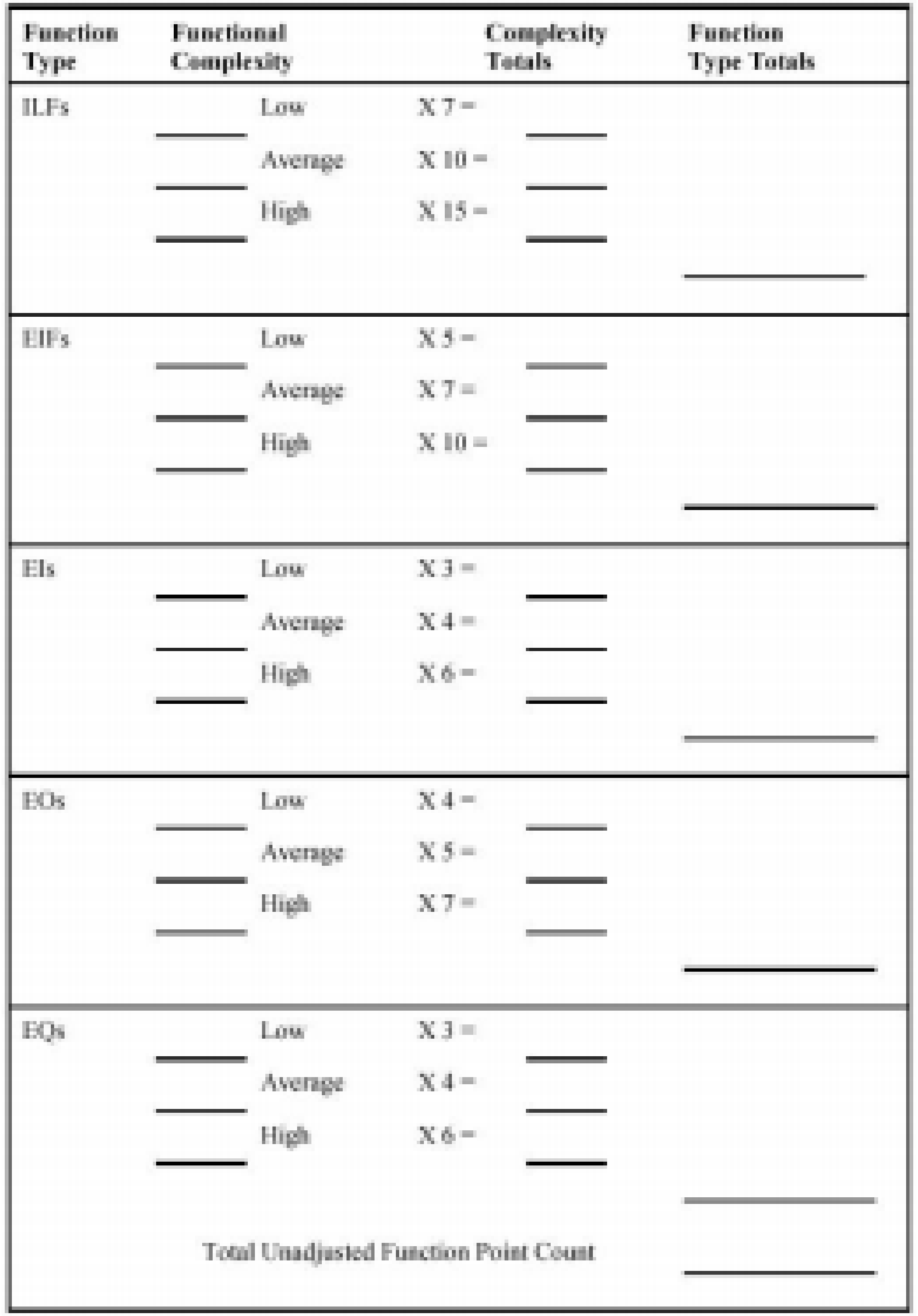
Size Metric: FUNCTION POINT ANALYSIS (FPA)

✓ **Based on the data function complexity weight is assigned to each data function**

Measurement Parameter	Low	Average	High
1. external inputs (EI)	7	10	15
2. external outputs (EO)	5	7	10
3. external inquiries (EQ)	3	4	6
4. internal files (ILF)	4	5	7
5. external interfaces (EIF)	3	4	6

✓ Count the number of all five components. The sum of each count is multiplied by an appropriate weight using Table above

✓ Add all the five results calculated in the previous step. This is the final **UFP.**



Function Type	Functional Complexity		Complexity Totals	Function Type Totals
ILFs	____ Low	X 7 =	____	
	____ Average	X 10 =	____	
	____ High	X 15 =	____	
				____
EIFs	____ Low	X 5 =	____	
	____ Average	X 7 =	____	
	____ High	X 10 =	____	
				____
EIs	____ Low	X 3 =	____	
	____ Average	X 4 =	____	
	____ High	X 6 =	____	
				____
EOs	____ Low	X 4 =	____	
	____ Average	X 5 =	____	
	____ High	X 7 =	____	
				____
EQs	____ Low	X 3 =	____	
	____ Average	X 4 =	____	
	____ High	X 6 =	____	
				____
	Total Unadjusted Function Point Count			____

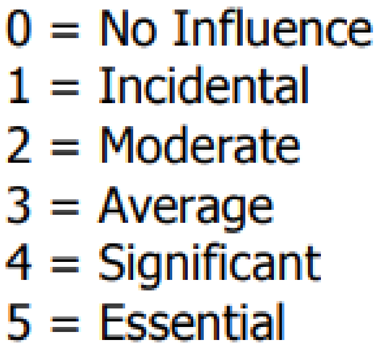
SW Size Estimation

Size Metric: FUNCTION POINT ANALYSIS (FPA)

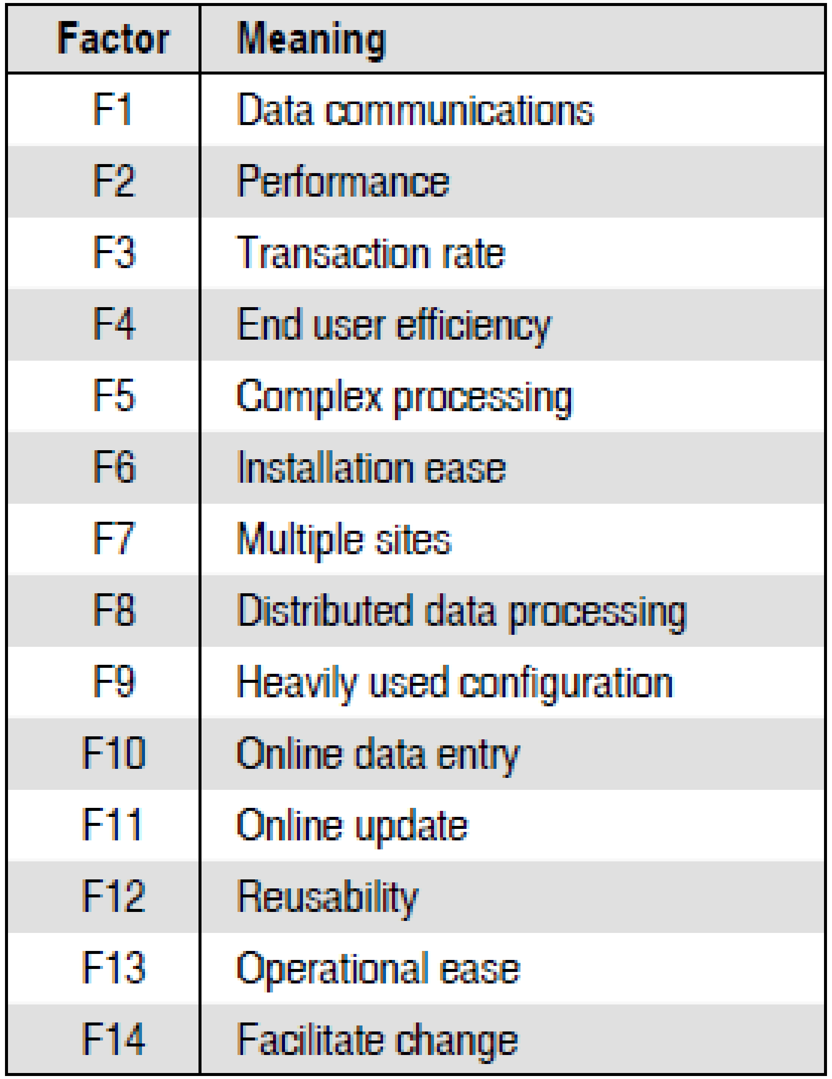
- Calculating **Adjusted Function Point**:
  - A **value adjustment factor (VAF)** is used as a multiplier of the unadjusted function point **(UFP)** count in order to calculate the adjusted function point **(AFP)** count of an application.
  - To calculate the VAF, we evaluate the 14 GSCs(General system Characteristics) on a scale of 0–5 to determine the degree of influence(DI) for each GSC description.
  - Add the DIs for all 14 GSCs to produce the **total degree of influence (TDI).**
  - Use the TDI in the following equation to compute VAF.

    VAF = (TDI × 0.01) + 0.065
  - The final adjusted function point is calculated as,

    AFP = UFP × VAF



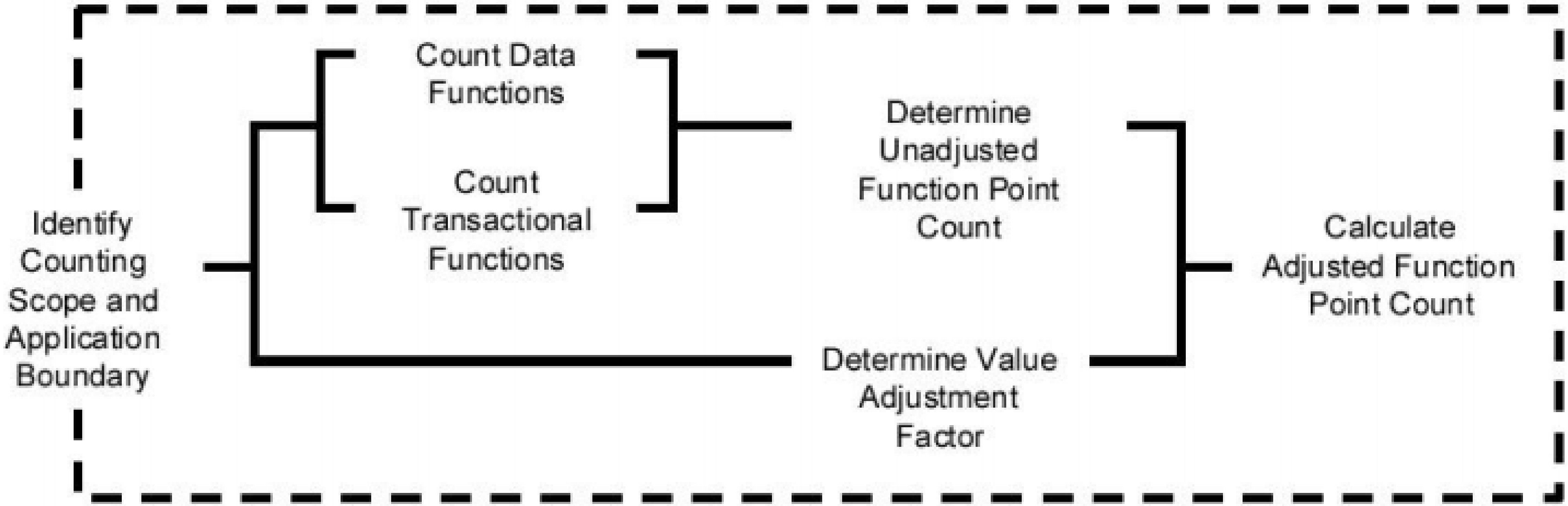
0 = No Influence
1 = Incidental
2 = Moderate
3 = Average
4 = Significant
5 = Essential



Factor	Meaning
F1	Data communications
F2	Performance
F3	Transaction rate
F4	End user efficiency
F5	Complex processing
F6	Installation ease
F7	Multiple sites
F8	Distributed data processing
F9	Heavily used configuration
F10	Online data entry
F11	Online update
F12	Reusability
F13	Operational ease
F14	Facilitate change

SW Size Estimation

FUNCTION POINT ANALYSIS (FPA)



Identify Counting Scope and Application Boundary
Count Data Functions
Count Transactional Functions
Determine Unadjusted Function Point Count
Determine Value Adjustment Factor
Calculate Adjusted Function Point Count

SW SIZE ESTIMATION

FUNCTION POINT ANALYSIS (FPA)

✓ Example:

Consider a project with the following parameters: EI = 50, EO = 40, EQ = 35, ILF = 06, and ELF = 04. Assume all weighing factors are **average**. In addition, the system requires **Significant** performance, **average** end-user efficiency, **moderate** distributed data processing, and **Significant** data communication. Other GSCs are **incidental**. Compute the function points using FPA.

SW Development Life Cycle (SDLC)

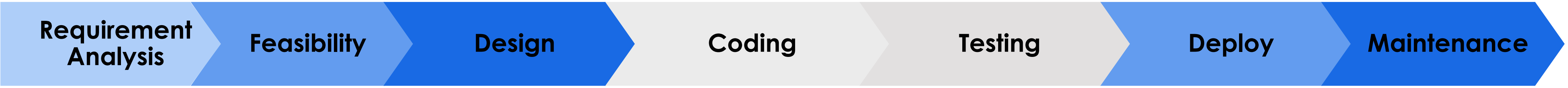
- ✓ is a systematic process for building software that ensures the **quality** and **correctness** of the software built.

- ✓ SDLC aims to produce **high-quality** software that meets **customer expectations**.

Why SDLC

- ✓ It offers a basis for project planning, scheduling, and estimating
- ✓ Provides a framework for a standard set of activities and deliverables
- ✓ It is a mechanism for project tracking and control
- ✓ Increases visibility of project planning to all involved **stakeholders** of the development process
- ✓ Increased and enhance development speed
- ✓ Improved client relations
- ✓ Helps you to decrease project risk and project management plan overhead

SDLC Phases



Requirement Analysis

Feasibility

Design

Coding

Testing

Deploy

Maintenance

Phase 1: Requirement gathering and analysis

Phase 2: Feasibility

Phase 3: Design

Phase 4: Coding

Phase 5: Testing

Phase 6: Installation/Deployment

Phase 7: Maintenance

Requirement gathering and analysis

- ✓ is one of the most critical phases of SDLC.
- ✓ the basis of the whole software development process.
- ✓ All the business requirements are gathered from the client in this phase.
- ✓ Requirements are analyzed to prepare SW specification, A well defined document known as Software Requirement Specification (SRS).

is divided into two phases where **requirement gathering** is related to identify client’s projects technical requisites and the **Analysis** is to evaluate whether the accumulated information and facts are valuable enough to proceed with the project plan.

REQUIREMENT GATHERING AND ANALYSIS

✓ Why is the requirement gathering important?

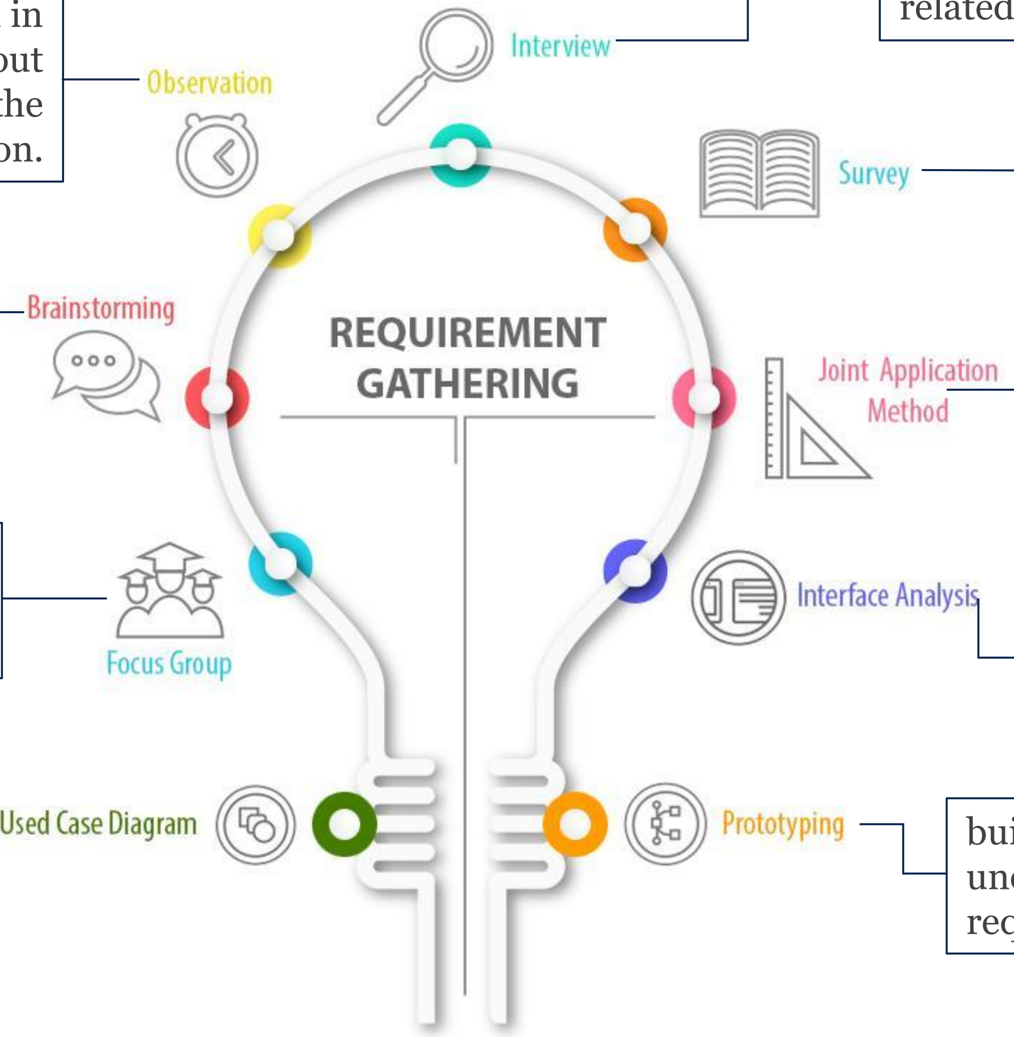
- ➢ is all about eliciting project requisites from client end so that they can get a full-fledged product embedded with all the specifications.
- ➢ it helps all the project associates communicate effortlessly with each other over defined clients needs.

✓ Who is responsible for requirements gathering?

- ➢ Business Analysts are the professionals who efficiently carry out software requirement gathering by breaking down the critical technical specifications into effective documentation and user stories.

REQUIREMENT GATHERING AND ANALYSIS

✔ **What are the tools and techniques of requirement gathering?**



the responsible person observes the team in working environment and gets ideas about the software and subsequently document the observation.

subject matter experts are the responsible people to conduct brainstorming sessions. They discuss and find out solutions to complex issues.

the idea is to collect information from representatives of clients and users to understand the software idea clearly.

Use case diagram is a technique that shows how people interact with software. It shows what a system does.

the business analyst talks to the user and clients who are unable to give out detailed information as they are not aware of the system development and related functionalities.

Questionnaire delivered to the user as well as the stakeholder for answers.

all the stakeholders like developers, end users, business analysts and software engineers come together and attend workshops for working on a system in greater detail.

Interface analysis is a specialized technique in which specific requirements related to application development are determined and their interaction with other software components is measured.

building a model of software which helps in uncovering and capturing software requirements from client.

REQUIREMENT GATHERING AND ANALYSIS

Challenges in Requirements Gathering

Lack of Clarity in Defining Criteria for Success

Clients Change their Minds Often

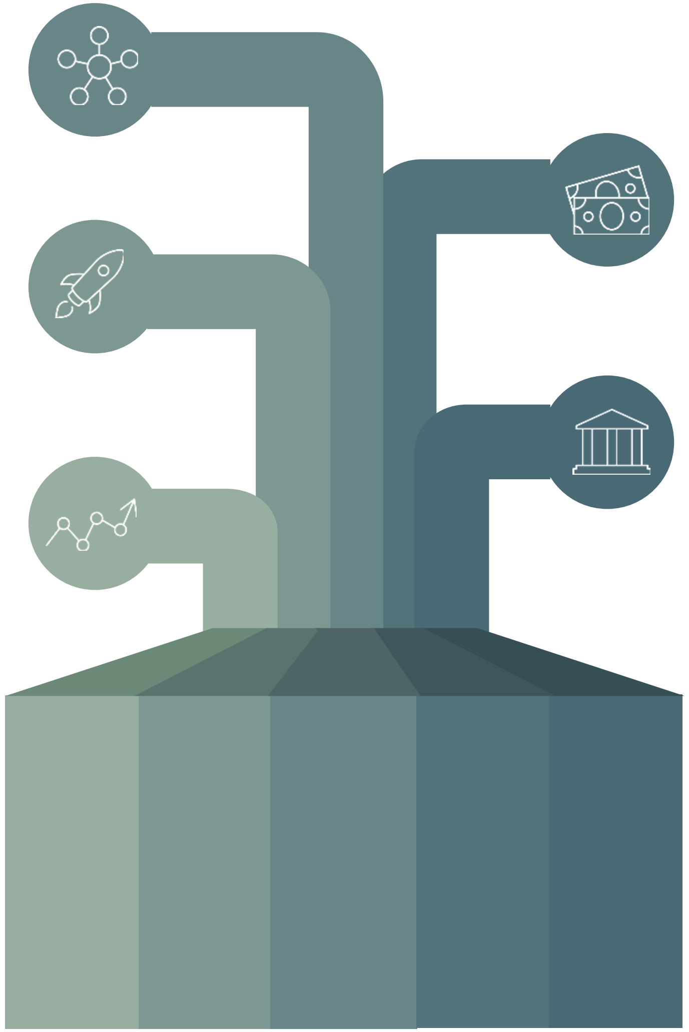
Lack of or Over Communication by Clients

Clients get stuck over certain Techniques/Solutions

Stakeholders can have Conflicting Priorities

Feasibility

Five types of feasibility check we need to perform



Operational 03

Can we create operations which is expected by the client?

Technical 04

Need to check whether the current computer system can support the software

Schedule 05

Decide that the project can be completed within the given schedule or not.

Economic 01

Can we complete the project within the budget or not?

Legal 02

Can we handle this project as cyber law and other regulatory framework/compliances.

DESIGN

- ✓ Design documents are prepared as per the SRS.
- ✓ Helps to define overall system architecture.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- ✓ Brief description and name of each module
- ✓ An outline about the functionality of every module
- ✓ Interface relationship and dependencies between modules
- ✓ Database tables identified along with their key elements
- ✓ Complete architecture diagrams along with technology details

Low-Level Design(LLD) / Detail Level Design (DLD)

- ✓ Functional logic of the modules
- ✓ Database tables, which include type and size
- ✓ Complete detail of the interface
- ✓ Addresses all types of dependency issues
- ✓ Listing of error messages
- ✓ Complete input and outputs for every module

DESIGN

Why software design is important?

Modularity
When you have a modular software, it's easy to move or even remove things if needed and even share the design work between multiple developers per software component.

Performance
design should tell how your system works internally and how it uses resources such as threads, database connections and other things that might make a hit to the performance.

Usability
design works as a great starting point for new-comers. Instead of making them look through all the source files you can just point them to the software design where they find all information they'll need to get started.

Maintainability
With good software design it's easier to maintain the software. You can spot straight away from the design how much a bug fix or introduction of a new feature will change the existing code base.

Portability
By including dependencies to other software modules, such as 3rd party libraries, in to your design makes the software much easier to port into another environment.

Trackability
Good design tracks the requirements and proves on the design level that all that is required from a piece of software is actually there..

CODING

- ✓ developers start build the entire system by writing code using the chosen programming language.
- ✓ tasks are divided into units or modules and assigned to the various developers.
- ✓ is the longest phase of the Software Development Life Cycle process
- ✓ developer needs to follow certain predefined coding guidelines
- ✓ also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

CODING

Things which we need to take care in coding phase:

1. Version control
2. Before actual coding, you must spend sometime to select dev tool which will be suitable for debugging, coding, modification.
3. before actual writing code, some standard should be defined, as multiple developers going to use the same file for coding
4. during development, developer should write appropriate comments so that others will come to know the logic behind the code
5. there should be a regular review meeting need to conduct in this stage to identify the prospective defects in an early stage.

TESTING

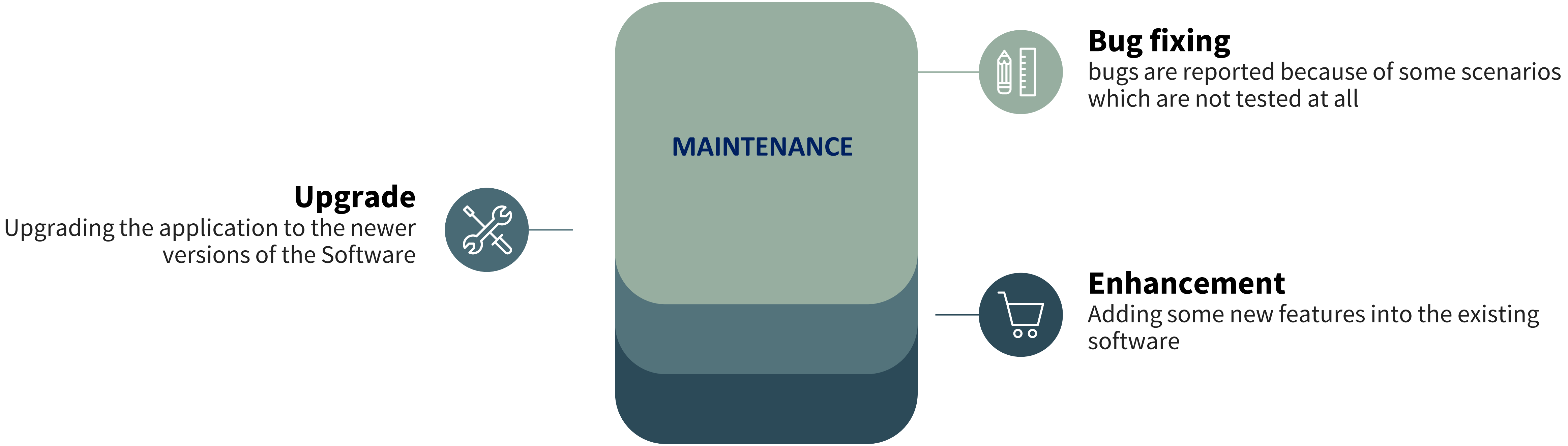
- ✓ Focus on investigation and discovery
- ✓ defines how the SW will be tested and who will perform
- ✓ find whether the developed code work according to the customer requirement.
- ✓ QA and testing team may find some bugs/defects which they communicate to developers.
- ✓ development team fixes the bug and send back to QA for a re-test.
- ✓ process continues until the software is bug-free, stable, and working according to the business needs of that system.

INSTALLATION/DEPLOYMENT

- ✓ Deployment is the process of promoting a fully-tested and approved application
- ✓ QA manager finalized the final bug-free release items.
- ✓ Deployment guide is provided to the operation team for hassle free deployment
- ✓ Deployment could require different promotion processes
  - ✓ For web applications, it would require promoting the application to a server, testing, redirecting DNS
  - ✓ For standalone applications, it would require producing media or downloadable packages
  - ✓ For software as a service, there are licensing issues that need to be determined
- ✓ Ensure data migration to the live server has occurred completely

MAINTENANCE

Major Activities after SW deployment/while SW in use by the end users



MAINTENANCE

Bug fixing
bugs are reported because of some scenarios which are not tested at all

Upgrade
Upgrading the application to the newer versions of the Software

Enhancement
Adding some new features into the existing software

The main focus of this phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.