GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: B. Tech

Course Code:BTCS3503

Course Name: Software Testing and Quality Assurance

Course Outcomes :

C01	**Understand and analyse the advantages of various Software Development Lifecycle Models**
C02	Explain and validate the various software testing activities.
C03	Execute the testing techniques for analysis.
C04	Apply and use different types of software testing tools and Select and understand the approaches used for the maintenance of a software.

Syllabus

- Software Crisis, Software Processes & Characteristics, Software life cycle models, Waterfall, Prototype, Evolutionary and Spiral Models,Requirement engineering, Use case approach, requirements analysis using DFD, Data dictionaries & ER Diagrams,Cohesion & Coupling What is software testing and why it is so hard, Error, Fault, Failure, Incident, Test Cases, Testing Process, Limitations of Testing, Unit Testing, Levels of Testing, Integration Testing, System Testing

•Debugging, Domain Testing, Path testing, DD-Paths, Cyclomatic Complexity, Graph Metrics, Data Flow Testing, Mutation testing, Grey Box Testing- types – Integration-System-Acceptance-Top-down-bottom up testing approach. Boundary value analysis, Equivalence class testing, Decision table-based testing, Cause effect graphing technique. Static Testing Tools, Dynamic Testing Tools, Characteristics of Modern Tools – Automated testing toolsManagement of Maintenance, Maintenance Process, Maintenance Models

- Reverse Engineering, Software Re-engineering, Configuration Management, Documentation. Software Quality, Software Control, Quality Assurance, Quality Assurance Analyst, Quality Factor, Quality Management, Methods of Quality Management, Core components of Quality, Cost Aspect of Quality. Quality Planning, Quality plan objectives, planning process overview, Business Plan and Quality Plan, TQM (Total Quality Management), TQM concepts, Zero defect movement Quality Models/Standards, Standards and guidelines, Types of ModelsI

Recommended Books

Text books

- Software Engineering: A practitioner's Approach, Roger S Pressman, Sixth Edition. McGraw-Hill International Edition, 2005.
- Software Engineering: Ian Summerville, Seventh Edition, Pearson Education, 2004.
- Daniel Galin, "Software Quality Assurance", Pearson Publication, 2009.

Reference Book

- Fundamentals of Software Engineering: Rajib Mall, PHI, 2005.
- Software Engineering, A Precise Approach, Pankaj Jalote, Wiley India,2010.
- Software Engineering: A Primer, Waman S Jawadekar, Tata McGraw-Hill, 2008.

Additional online materials

Software Components

- Software Components are parts of a system or applications. Components are a means of breaking the complexity of software into manageable parts.
- Each component hides the complexity of its implementation behind an interface.
- Components can be swapped in and out like interchangeable parts of a machine.

Software Components

- ➢ This reduces the complexity of software development, maintenance, operations and support and allows the same code to be reused in many places.
- ➢ The following are illustrative examples of a component.
- ❖ Views
- ❖ Models
- ❖ Controllers
- ❖ Data Access Objects
- ❖ Services
- ❖ Plugins
- ❖ APIs

Software Components

1.Views

- User interface components for different requests, views and scenarios.
- For example, difficult components can be used to display the same information in a web page and mobile app.

Software Components

2. Models

- ➢Components that handle requests or events including business rules and data processing.
- ➢For example, a model might handle a bill payment request for an internet banking website.

Software Components

3.Controllers

- A controller is a component that decides what components to call for a particular request or event.
- For example, a controller might dynamically load different views for a bill payment on factors such as language, transaction status or channel.

Software Components

4.Data Access Objects

- A data access object provides an abstract interface for databases.
- In theory, this allows you to switch to a different database without the application needing to know. This shows the power of a component based approach as dramatic changes can be confined to a relatively small section of a code base.

Software Components

5. Services

- A service is a component that is deployed independently.
- For example, a bank might deploy a market data service to cloud infrastructure. This service would provide stock market data to a variety of stock trading system and applications.
- Services allow for extremely resilient applications.
- For example, if an application doesn't get a response from a service, it can try again and be directed to a completely different instance.

Software Components

6. Plugins

- ➢Components designed to extend the functionality of an application or system.
- ➢For example, a plugin for a media player to visualize music.

Software Components

7.APIs

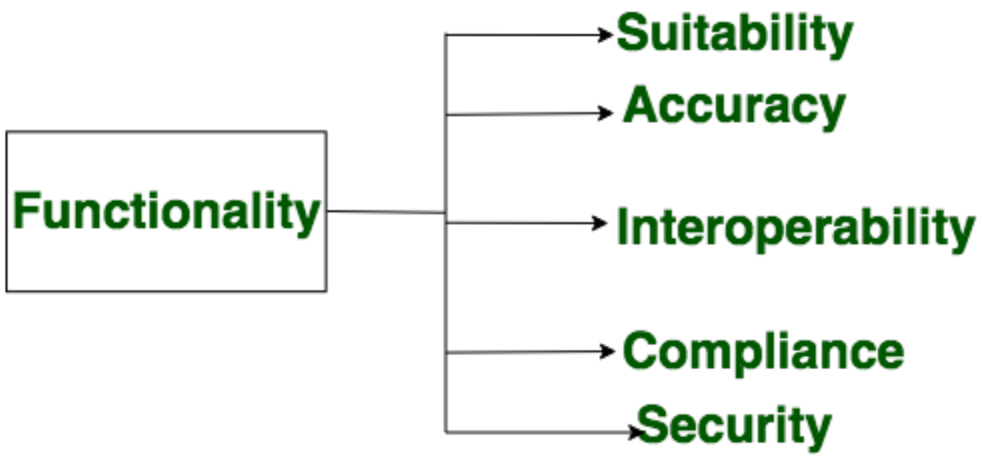
- A component that can be reused across multiple systems and applications can be packaged and distributed as an API.
- For example, an open source API to connect to a particular database.

Software Characteristics

- **Software** is defined as collection of computer programs, procedures, rules and data.
- Software Characteristics are classified into six major components:
  - Functionality
  - Reliability
  - Efficiency
  - Usability
  - Maintainability
  - Portability

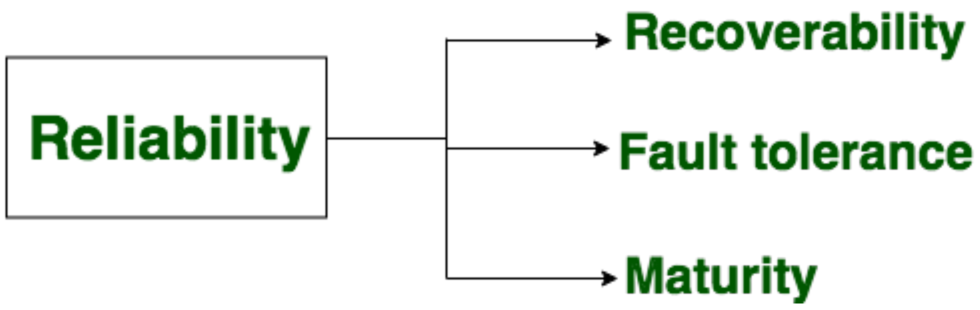
Software Characteristics 1.Functionality

- It refers to the degree of performance of the software against its intended purpose. Required functions are:



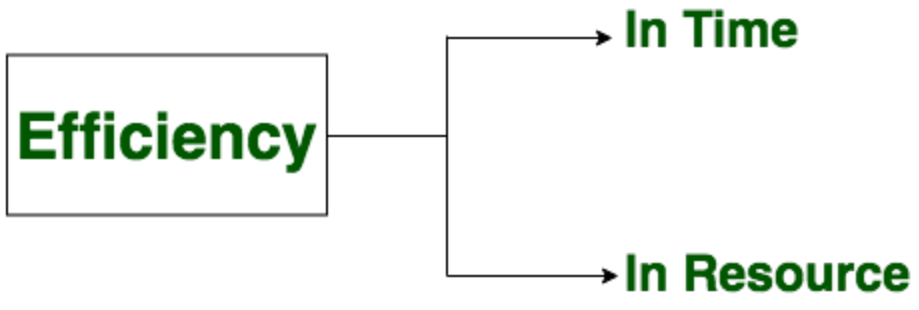
Software Characteristics
2.Reliability

- A set of attribute that bear on capability of software to maintain its level of performance under the given condition for a stated period of time. Required functions are:



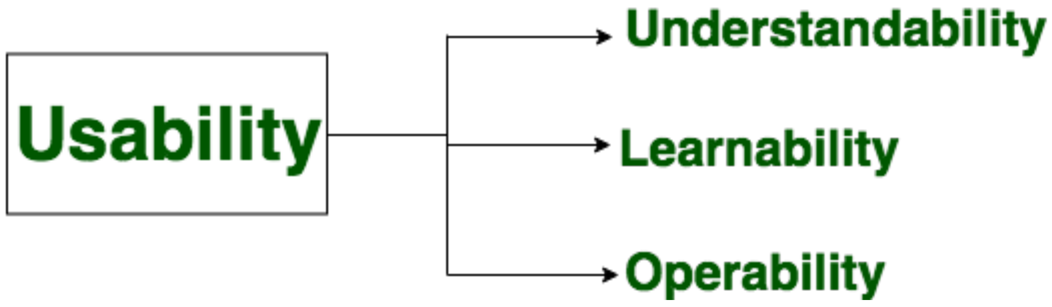
Software Characteristics
3.Efficiency

- It refers to the ability of the software to use system resources in the most effective and efficient manner.
- the software should make effective use of storage space and executive command as per desired timing requirement.
- Required functions are:



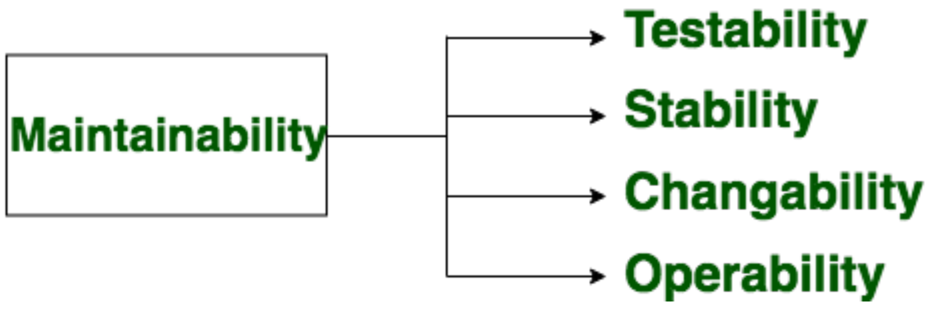
Software Characteristics
4.Usability

- It refers to the extent to which the software can be used with ease.
- the amount of effort or time required to learn how to use the software.



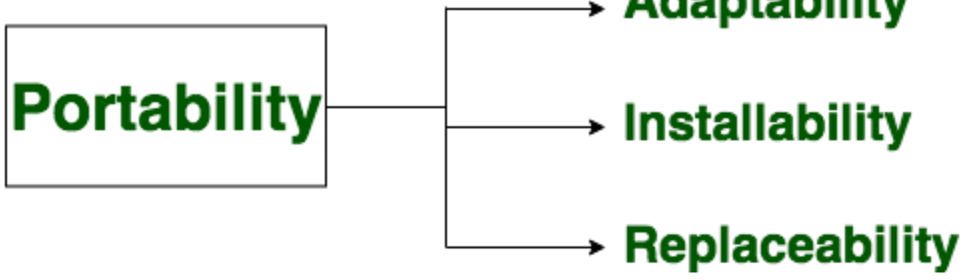
Software Characteristics
5.Maintainability

- It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.
- Required functions are:



Software Characteristics
6.Portability

- A set of attribute that bear on the ability of software to be transferred from one environment to another, without or minimum changes.
- Required functions



Software Applications

- System Software
- Application Software
- Engineering / Scientific Software
- Embedded Software
- Product-line Software
- Web Apps (Web Applications)
- AI Software

SOFTWARE CRISIS

SOFTWARE CRISIS

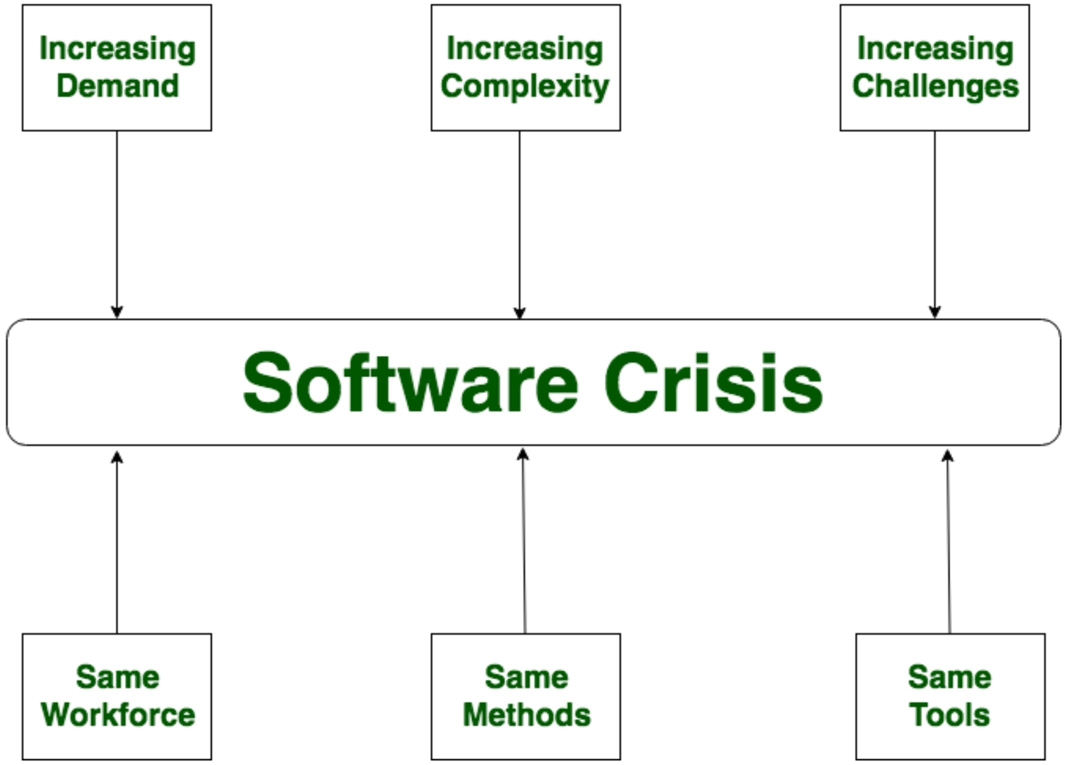
Size: Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.

Quality: Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.

Cost: Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.

SOFTWARE CRISIS

Delayed Delivery: Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.



These are not isolated incidents:

IBM survey of 24 companies developing distributed systems:

- ❖ 55% of the projects cost more than expected
- ❖ 68% overran their schedules
- ❖ 88% had to be substantially redesigned

Solution of Software Crisis

There is no single solution to the crisis. one possible solution of software crisis is *Software Engineering* because software engineering is a systematic, disciplined and quantifiable approach. For preventing software crisis, there are some guidelines:

- ✓ Reduction in software over-budget
- ✓ The quality of software must be high
- ✓ Less time needed for software project
- ✓ Experience working team member on software project
- ✓ Software must be delivered

Software Engineering Process

Software Engineering Process

- A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities.
- **Software specifications:** The functionality of the software and constraints on its operation must be defined.
- **Software development:** The software to meet the requirement must be produced.
- **Software validation:** The software must be validated to ensure that it does what the customer wants.
- **Software evolution:** The software must evolve to meet changing client needs.

Software process:

Software engineering process is the glue that holds:

- Technology together
- enables rational and timely development of computer software.

Software engineering process is a framework of a set of key process areas.

Large software systems often:

- ❖ Do not provide the desired functionality
- ❖ Take too long to build
- ❖ Cost too much to build
- ❖ Require too much resources (time, space) to run
- ❖ Cannot evolve to meet changing needs

- For every 6 large software projects that become operational, 2 of them are canceled

- On the average software development projects overshoot their schedule by half

- 3 quarters of the large systems do not provide required functionality

It forms a basis for:

- project management, budget and schedule control.
- applications of technical methods
- product quality control
- Identify new problems and solutions in software production.
- Study new systematic methods, principles, approaches for system analysis, design, implementation, testing and maintenance.
- Provide new ways to control, manage, and monitor software process.
- Build new software tools and environment to support software engineering.

Major Goals:

- ❖ To increase software **productivity** and **quality**.
- ❖ To effectively control software **schedule** and planning.
- ❖ To reduce the **cost** of software development.
- ❖ To meet the **customers'** needs and requirements.
- ❖ To enhance the conduction of software engineering **process**.
- ❖ To improve the current **software engineering practice**.
- ❖ To support the engineers' activities in a systematic and efficient manner.

Similarity and Differences from Conventional Engineering Processes

Programming versus Software Engineering

Programming

1. The process of translating a problem from its physical environment into a language that a computer can understand and obey. (*Webster's New World Dictionary of Computer Terms)*
2. The art of debugging a blank sheet of paper.

Software Engineering (according to Fritz Bauer)

"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

Difference between software engineering and computer science

Computer Science

Is concerned with

Software Engineering

- Theory
- Fundamentals

- Practical of Developing
- Delivering Useful software

Computer science theories are currently insufficient to act as a complete underpinning for software engineering, BUT it is a foundation for practical aspects of software engineering

Difference between software engineering and system engineering

Software engineering is part of **System engineering**

System engineering is concerned with all aspects of computer-based systems development including

- ❖ hardware, software and process engineering

System engineers are involved in

- ❖ system specification, architectural design,
- ❖ integration and deployment

Software Quality Attributes

Software Quality Attributes

What is Quality?

❖ Quality is defined as the product or services that should be "fit for use and purpose."

❖ Quality is all about meeting the needs and expectations of customers concerning functionality, design, reliability, durability, and price of the product.

Software Quality Attributes

- **How a Product Developer will define Quality?**

- The product meets customer requirements.

  How Customer will define Quality?

- The required functionality is provided with user-friendly manner.

Factors to measure Software Development Quality.

- Each attribute can be used to measure product performance. These attributes can be used for Quality assurance as well as Quality control.
- **Quality Assurance** activities are oriented towards prevention of introduction of defects
- **Quality Control** activities are aimed at detecting defects in products and services.

Software Quality Attributes

1. Reliability
2. Maintainability
3. Usability
4. Correctness
5. Efficiency
6. Integrity or Security
7. Testability
8. Flexibility
9. Reusability
10. Interoperability

Software Quality Attributes

1. Reliability

✓ Measure if the product is reliable enough to sustain in any condition. Should give consistently correct results.

✓ Product reliability is measured in terms of working of the project under different working environment and different conditions.

2. Maintainability

✓ Different versions of the product should be easy to maintain. For development it should be easy to add code to the existing system, should be easy to upgrade for new features and new technologies from time to time.

✓ Maintenance should be cost-effective and easy. The system is easy to maintain and correcting defects or making a change in the software.

Software Quality Attributes

3. Usability

This can be measured in terms of ease of use. The application should be user-friendly. Should be easy to learn. Navigation should be simple.

The system must be:

- ➢ Easy to use for input preparation, operation, and interpretation of the output.
- ➢ Provide consistent user interface standards or conventions with our other frequently used systems.
- ➢ Easy for new or infrequent users to learn to use the system.

Software Quality Attributes

4. Portability

- ❖ This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioral issues related to porting.

5. Correctness

- ❖ The application should be correct in terms of its functionality, calculations used internally and the navigation should be correct.
- ❖ This means the application should adhere to functional requirements.

Software Quality Attributes

6. Efficiency

- ❖ Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example, the system should utilize processor capacity, disk space and memory efficiently.
- ❖ If system is using all the available resources then the user will get degraded performance failing the system for efficiency. If system is not efficient then it can not be used in real-time applications.

Software Quality Attributes

7. Integrity or Security

- Integrity comes with security. System integrity or security should be sufficient to prevent unauthorized access to system functions, preventing information loss
- Ensure that the software is protected from virus infection, and protecting the privacy of data entered into the system.

Software Quality Attributes

8. Testability

- ❖ The system should be easy to test and find defects. If required should be easy to divide into different modules for testing.

9. Flexibility

- ❖ Should be flexible enough to modify. Adaptable to other products with which it needs interaction. Should be easy to interface with other standard 3rd party components.

Software Quality Attributes

10. Reusability

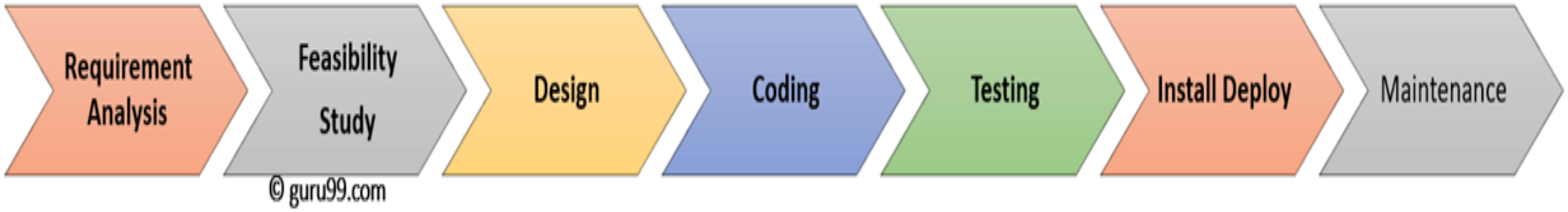
- ❖ Software reuse is a good cost-efficient and time-saving development way.
- ❖ Different code libraries classes should be generic enough to use easily in different application modules.
- ❖ Dividing application into different modules so that modules can be reused across the application.

Software Quality Attributes

11. Interoperability

- Interoperability of one system to another should be easy for the product to exchange data or services with other systems.
- Different system modules should work on different operating system platforms, different databases, and protocols conditions.
- Applying above quality attributes standards we can determine whether the system meets the requirements of quality or not.

SDLC Phases



Phase 1: Requirement collection and analysis
Phase 2: Feasibility study:
Phase 3: Design:
Phase 4: Coding:
Phase 5: Testing:
Phase 6: Installation/ Deployment:
Phase 7: Maintenance:

SDLC Models

Sequential model
- Waterfall model.
- V-model.

Incremental model
- Incremental model.
- Spiral model
- RAD(Rapid application development) model.

Waterfall Model

- ➢ It is also called as linear sequential model.
- ➢ In this model whole application is developed in a sequential approach.
- ➢ In this model each phase must be completed fully bef the next phase begin.
- ➢ Provides structure to inexperienced staff.

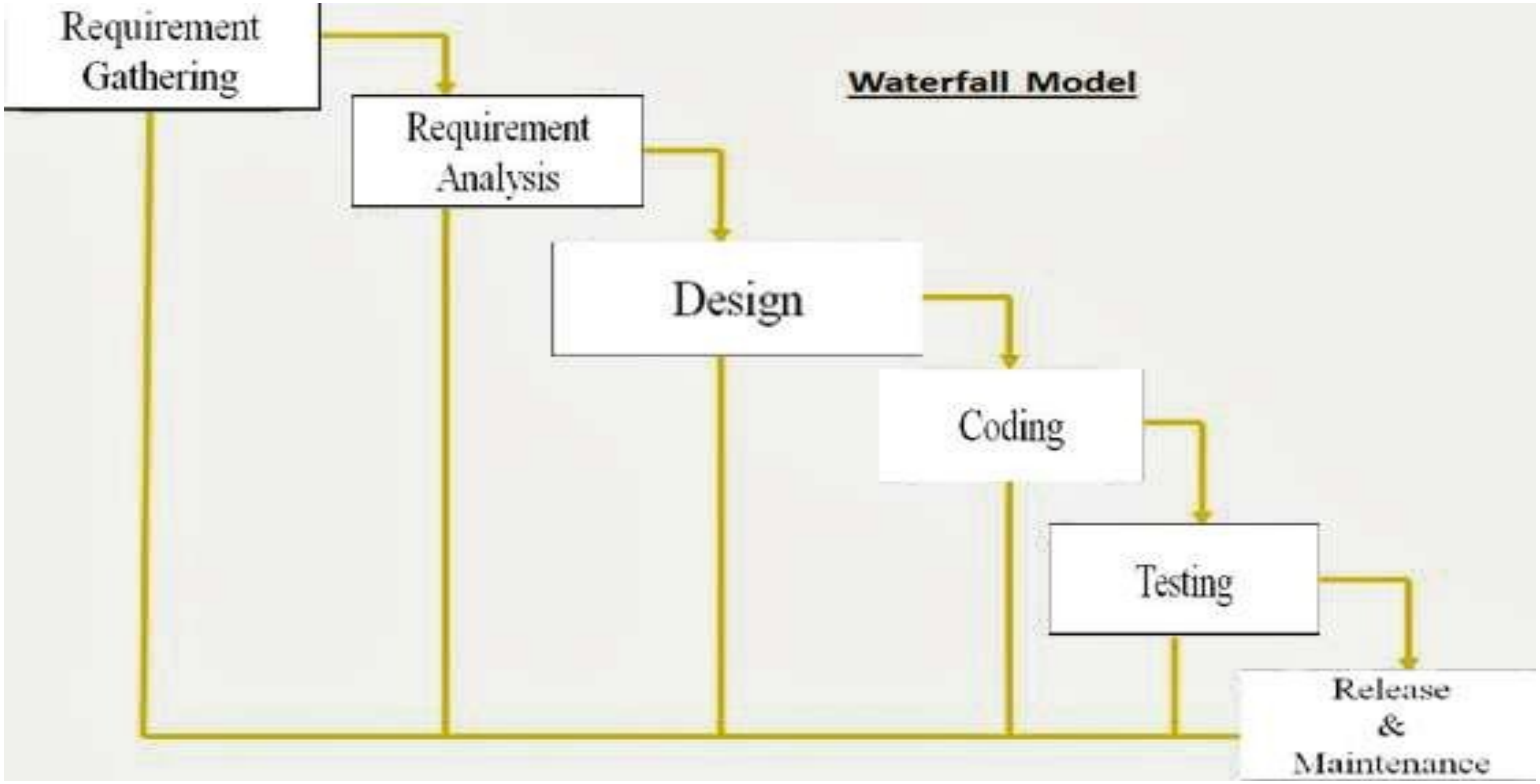
History of waterfall mode

- The first formal description of the waterfall model is often cited as a 1970 article by Winston W.Royce.
- Royce presented this model as as an example of a flawed,non-working model.
- It has been widely used for software projects ever since.

Where to use the waterfall model

- ➢ Requirements are very well known.
- ➢ Product definition is stable.
- ➢ Technology is understood.
- ➢ New version of an existing product

Waterfall model Diagram



Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation – With inputs from the system design, the system is first developed in small

Integration and Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product

ADVANTAGES

- A waterfall model is easy to implementation.
- It helps to find errors earlier
- Easy to understand, easy to use.
- Works well when quality is more important than cost or schedule
- Documentation is produced at every stage of a waterfall model allowing people to understand what has been done.

Disadvantages

- ➢ It is only suitable for the small size projects.
- ➢ Constant testing of the design is needed.
- ➢ If requirements may change the Waterfall model may not work.
- ➢ Difficult to estimate time and cost for each stage of the development process.
- ➢ Adjust scope during the life cycle can kill a project

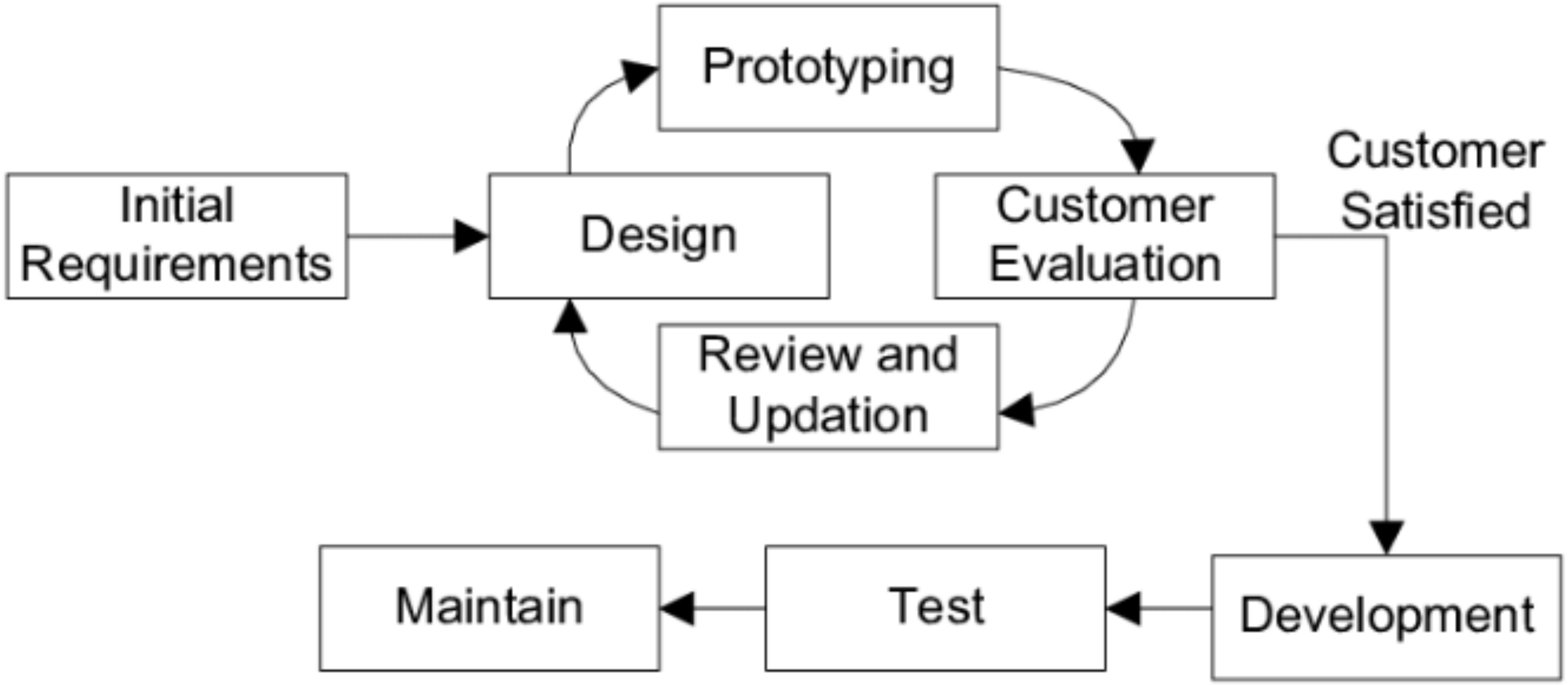
PROTOTYPE MODEL

- A prototyping model suggest that **before carrying out the development of the actual software, a working prototype of the system should be built.**
- A prototype is a toy implementation of the system.
- Prototype is a working model of software with some **limited functionality.**
- Prototyping is used to allow the users evaluate the developer proposals and try them out before implementation.
- By using this prototype, customer can understand the requirements of desired system and also the customer can get an **“actual feel” of the system**. It is an attractive idea for complex and bigger systems.

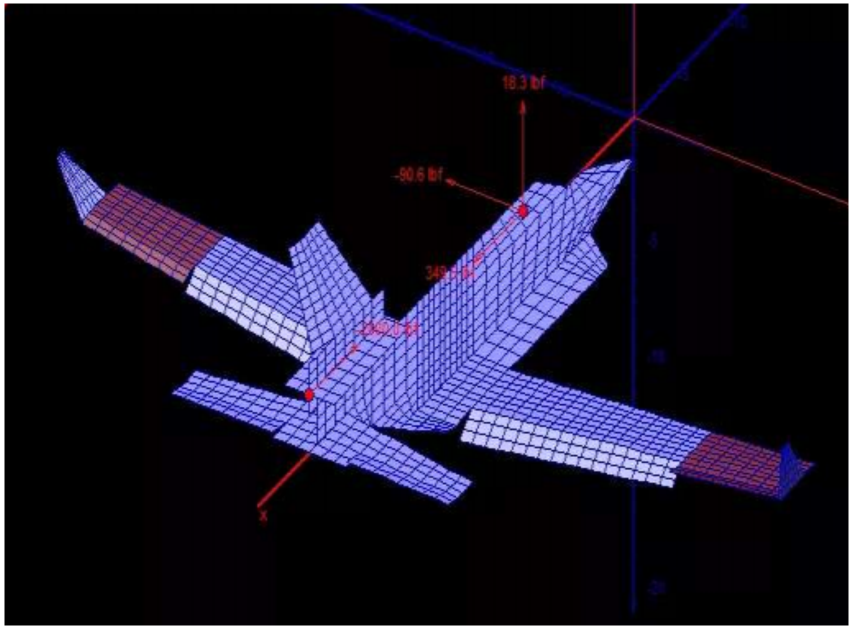
BEST PRACTICES OF PROTOTYPING

- 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

PROTOTYPE MODEL



PROTOTYPE MODEL



PROTOTYPE MODEL

Requirements gathering and Analysis

A prototype model begins with requirements analysis, and the requirements of the system are define in detail. The user is interviewed in order to know the requirements of the system.

Quick design

When requirements are known, a quick design for the system is created. It is not a detailed design , it includes the important aspects of the system, which gives an idea of the system to the user.

PROTOTYPE MODEL

Build prototype:

Information gathering from quick design is modified to form a prototype .It represents a ‘rough’ design of the required system.

Customer evaluation of prototype:

The build prototype is presented to the customer for his/her evaluation.

PROTOTYPE MODEL

Prototype refinement:

Once the user evaluate the prototype, it is refined according to the requirements .

When the user is satisfied to the developed prototype , a final system is developed based on the final prototype ,

which is developed by the iterative method means we design the system according to the final prototype , after that implement , test the product to find the error and at last we maintain the system.

PROTOTYPE MODEL

Advantages

- User early in the process , enabling early assessment and increasing the user confidence
- Better implementation of the requirements.
- It helps in reducing the risk associated to the project.
- Improving communication between developer and user

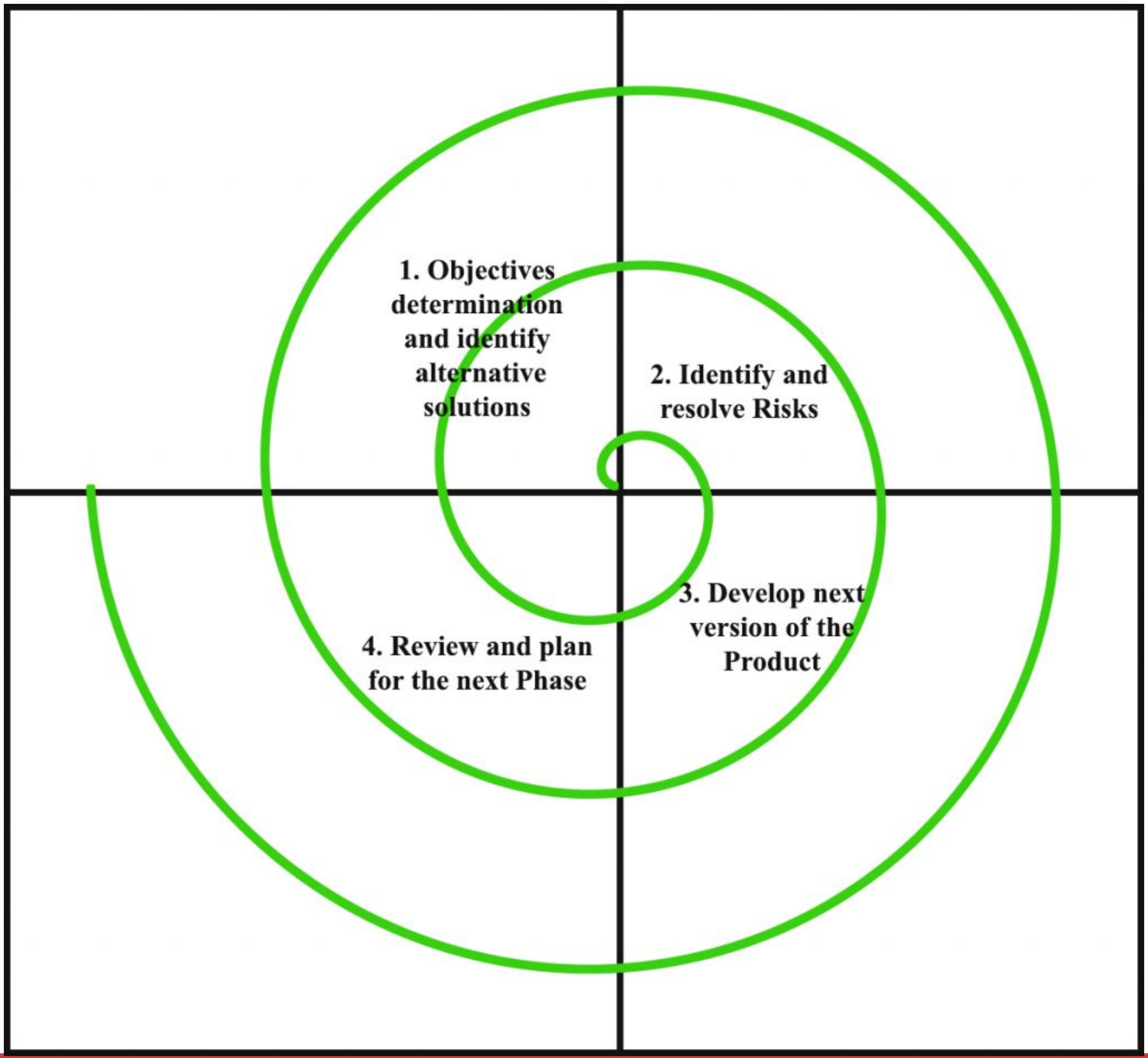
SPIRAL MODEL

- Each loop of the spiral is called a Phase of the software development process.

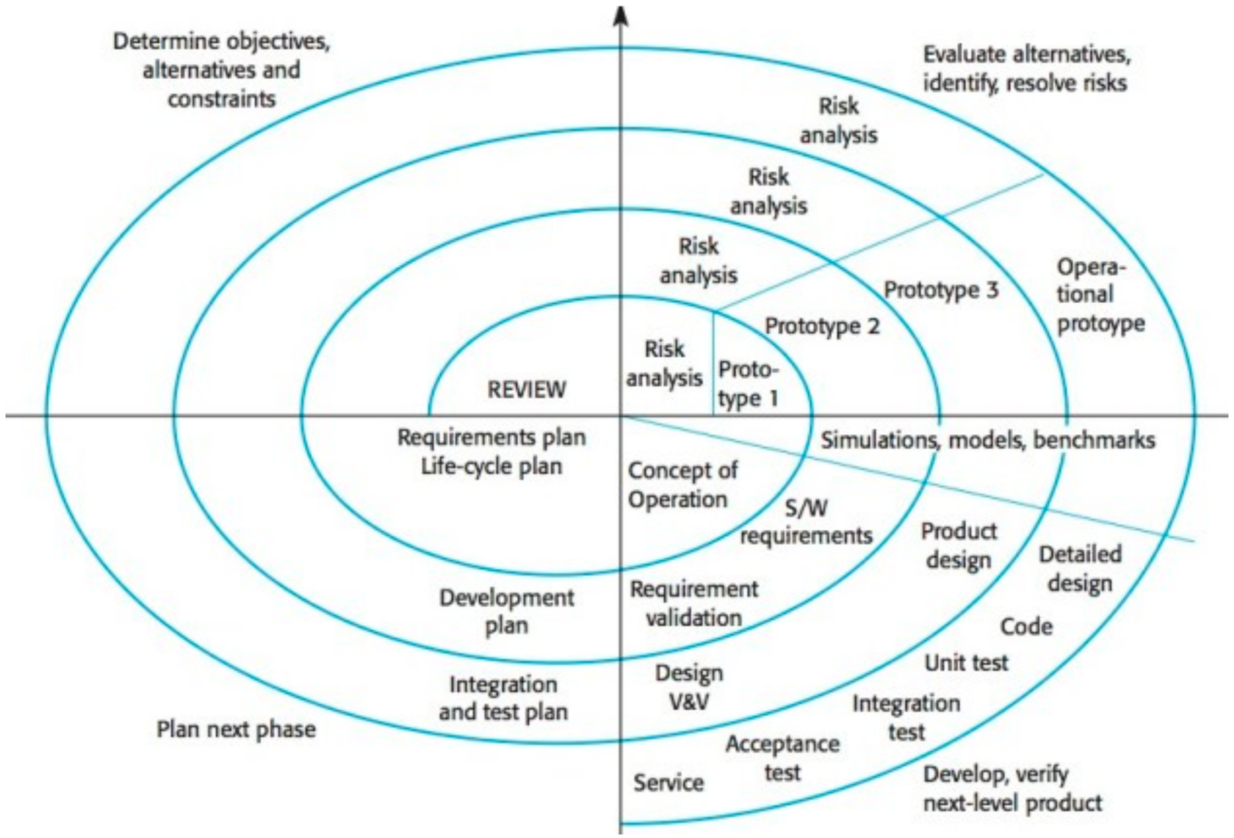
Two main distinguishing features:

- one is cyclic approach for incrementally growing a system’s degree of definition and implementation while decreasing its degree of risk
- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

SPIRAL MODEL


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Spiral Model



SPIRAL MODEL

Objectives determination and identify alternative solutions:

Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.

SPIRAL MODEL

Develop next version of the Product:

During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

Review and plan for the next Phase:

In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

SPIRAL MODEL

Important aspects

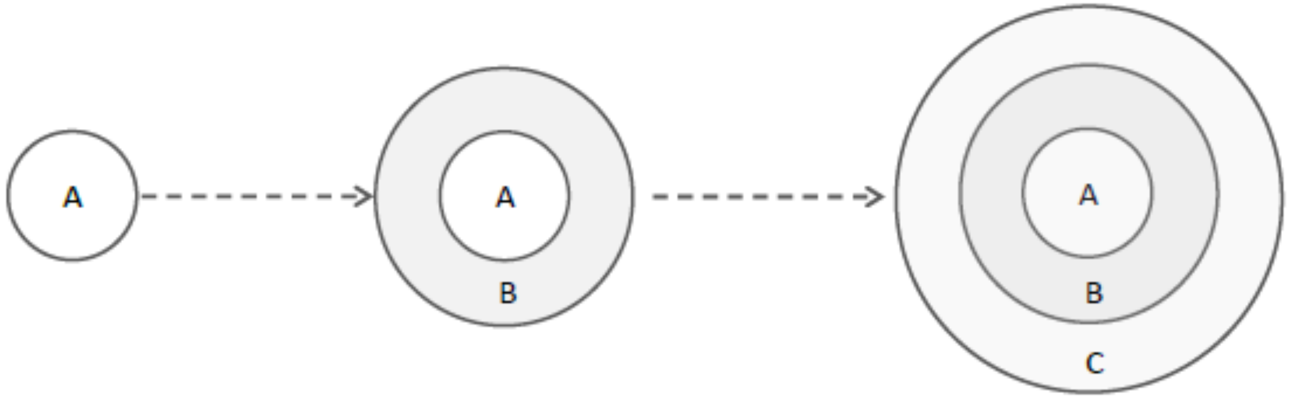
- Adjustments to the project plan
- Cost and schedule are adjusted based on feedback
- The number of iterations will be adjusted by project manager.

EVOLUTIONARY DEVELOPMENT MODEL

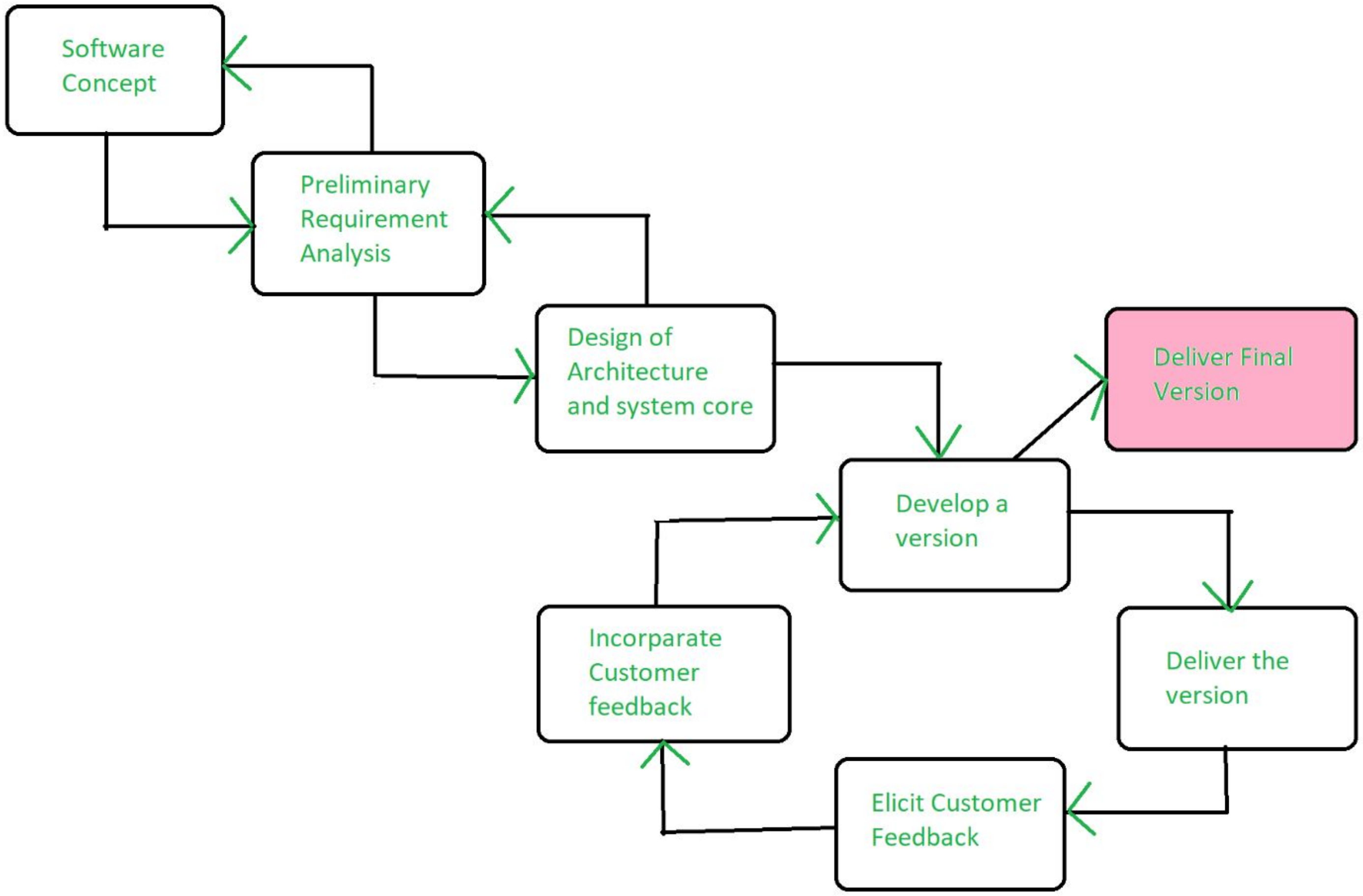
- *Evolutionary model is also known as **incremental model**.*
- In SDLC, Evolutionary Development Model the requirement is broken down into different functional units.
- These functional units can also be referred to as module.
- These modules can be incrementally built and delivered.
- Here at the beginning the core module of the software product is developed.
- New functionality is built, added to existing one and released as new version.
- Each successive version is capable of performing more functions in comparision to its previous versions.

EVOLUTIONARY DEVELOPMENT MODEL

- Each successive version/model of the product is a fully functioning software capable of performing more work than the previous versions/model.



EVOLUTIONARY DEVELOPMENT MODEL


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Develop a version
Deliver Final Version
Deliver the version
Elicit Customer Feedback
Incorparate Customer feedback

WHERE SDLC EVOLUTIONARY MODEL IS SUITABLE TO USE?

- Suitable for Object oriented software development product. Its because in OOP requirement can be separated in different modules in terms of the objects.
- Suitable for large products where requirements can be divided into modules.
- Every version will get built and delivered to customer who can then use it instead of waiting for the complete system.

ADVANTAGES OF EVOLUTIONARY MODEL

- Large project
- Chance to experiment with a partially developed software
- Elicit user requirements during the delivery of different versions of the software
- Reducing the chances of errors
- Avoids the need to commit large resources

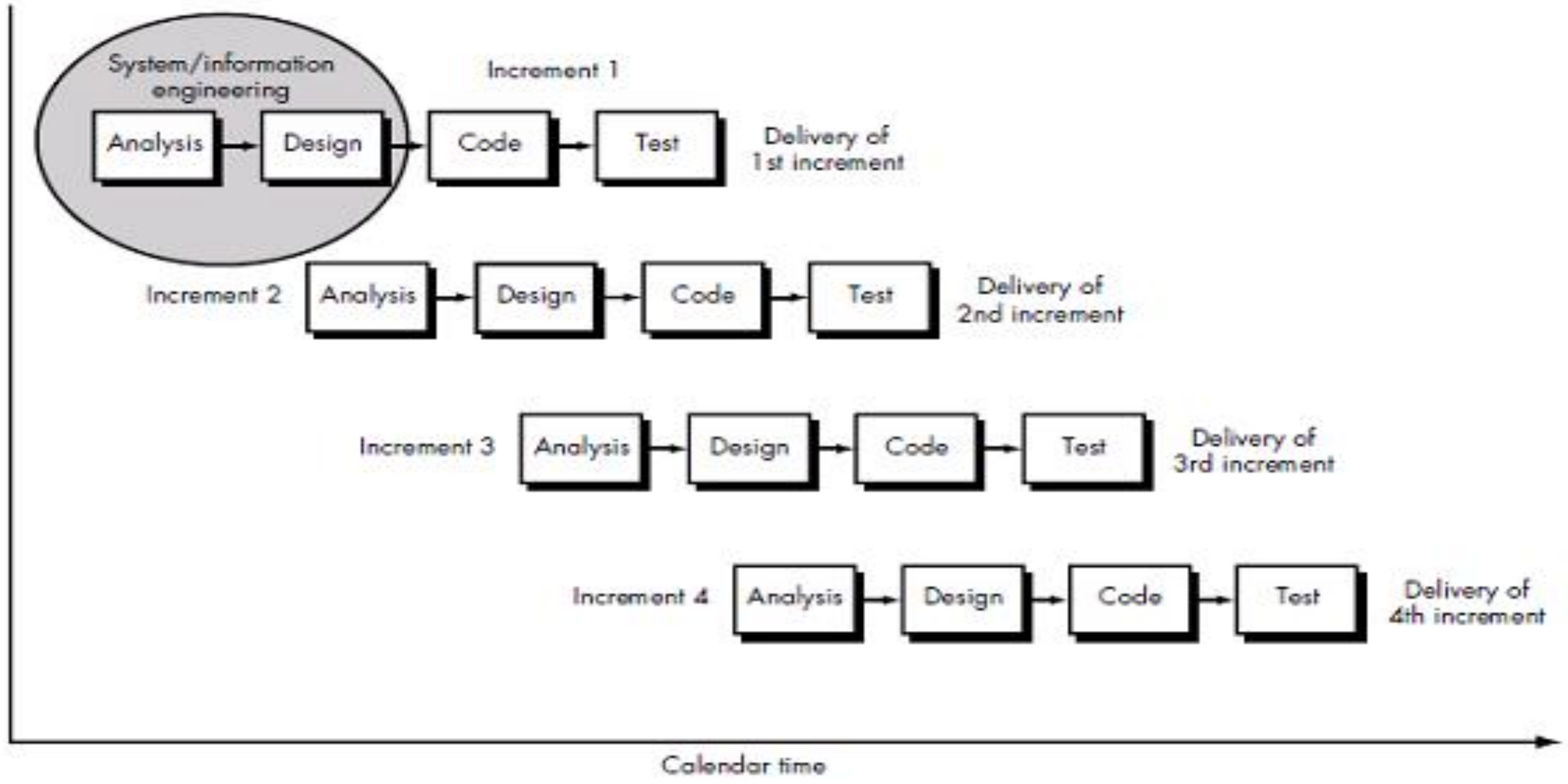
DISADVANTAGES OF EVOLUTIONARY MODEL

- Difficult to divide the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

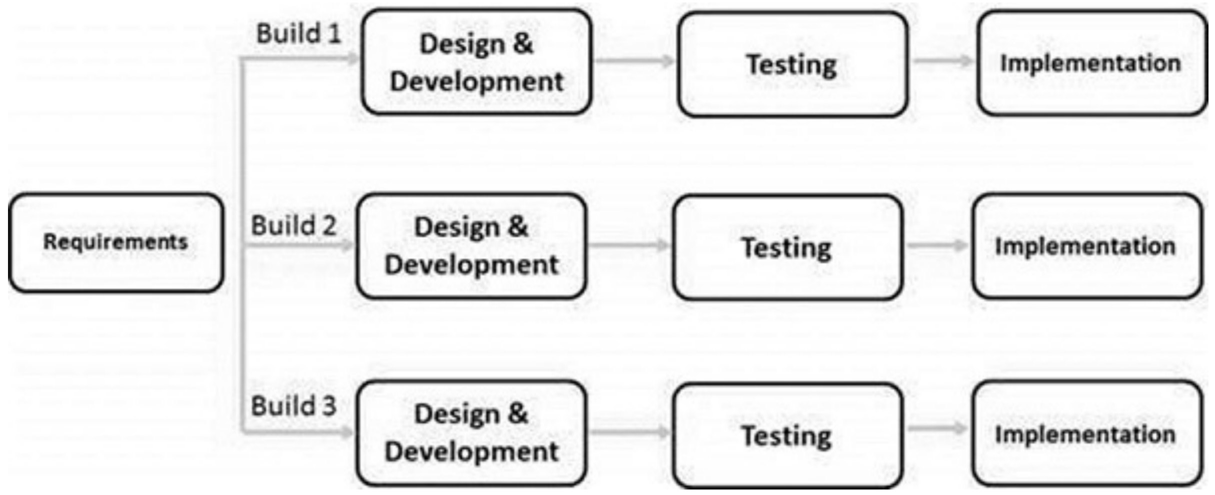
ITERATIVE ENHANCEMENT MODEL

- Iterative Enhancement Model also known as **Incremental Model**
- **Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.**
- The incremental model has phases similar to the linear sequential model arid has an iterative nature of prototyping.
- Difficult to divide the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

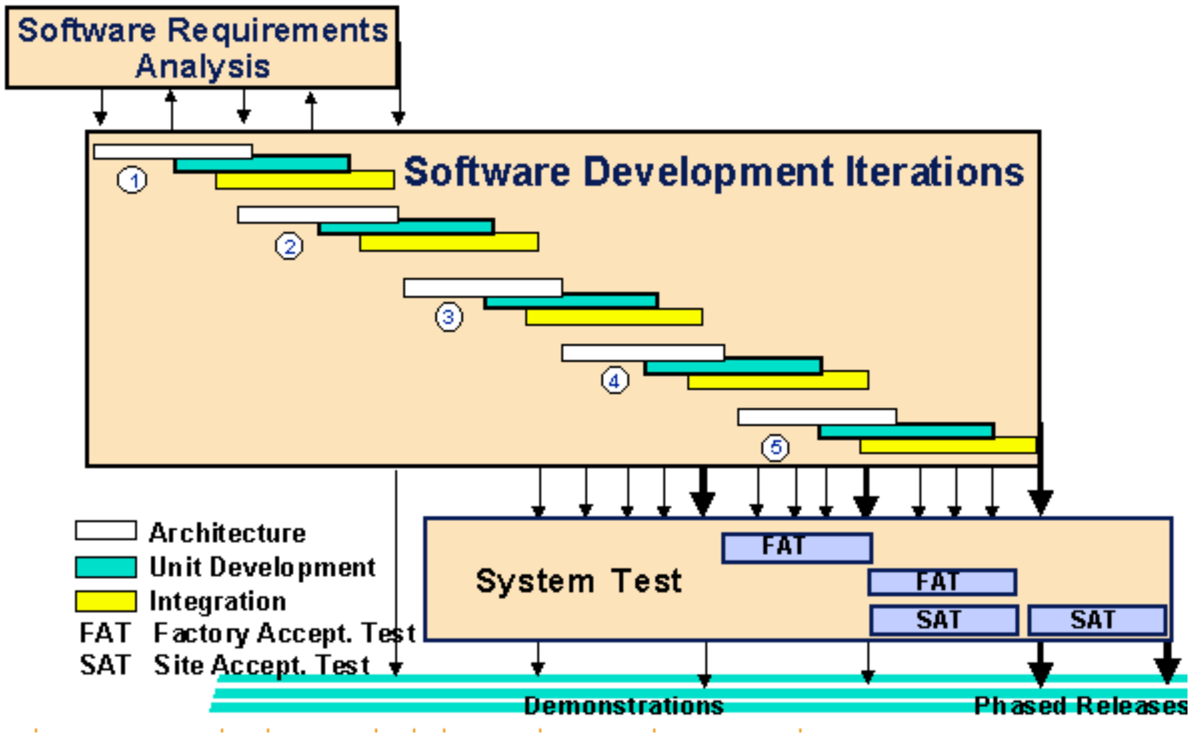
Iterative Enhancement Model



Iterative Enhancement Model



Iterative Enhancement Model



WHEN TO USE INCREMENTAL MODELS?

- Requirements of the system are clearly understood
- When demand for an early release of a product arises
- When software engineering team are not very well skilled or trained
- When high-risk features and goals are involved
- Such methodology is more in use for web application and product based companies

ADVANTAGES OF ITERATIVE ENHANCEMENT MODEL

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

TOPICS TO BE COVERED

Low Level Design:

Modularization,

Design Structure Charts,

Pseudo Codes,

MODULARIZATION

- Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently.

- Designers tend to design modules such that they can be executed and/or compiled separately and independently.

ADVANTAGE OF MODULARIZATION

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

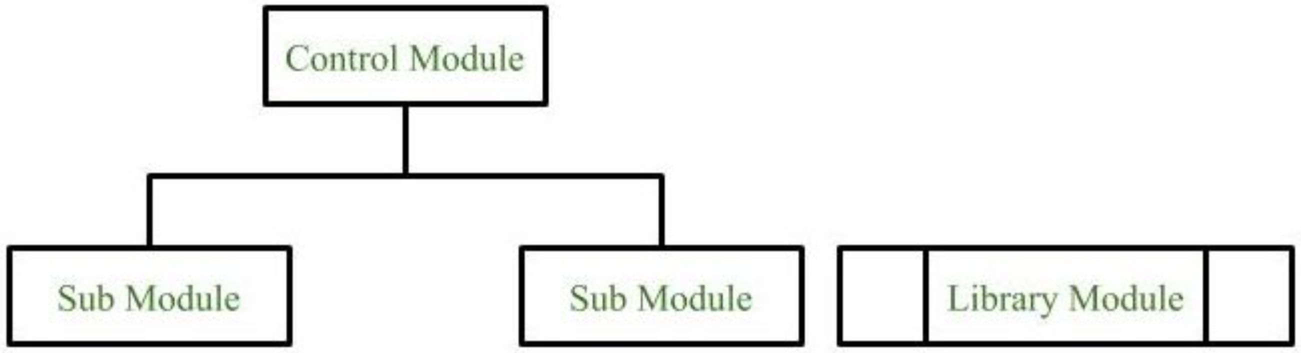
STRUCTURE CHART

- **Structure Chart** represent hierarchical structure of modules.
- It breaks down the entire system into lowest functional modules, describe functions and sub-functions of each module of a system to a greater detail.

SYMBOLS USED IN CONSTRUCTION OF STRUCTURED CHART

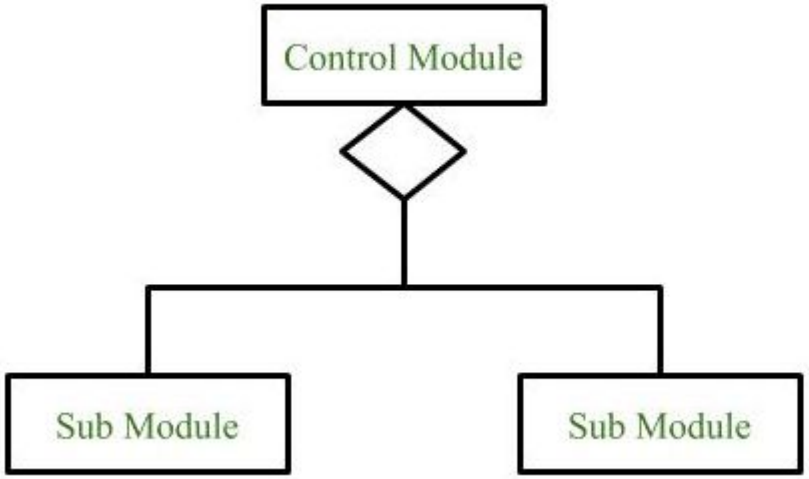
- **Module**
  It represents the process or task of the system. It is of three types.
  - **Control Module**
    A control module branches to more than one sub module.
  - **Sub Module**
    Sub Module is a module which is the part (Child) of another module.
  - **Library Module**
    Library Module are reusable and invokable from any module.

MODULE



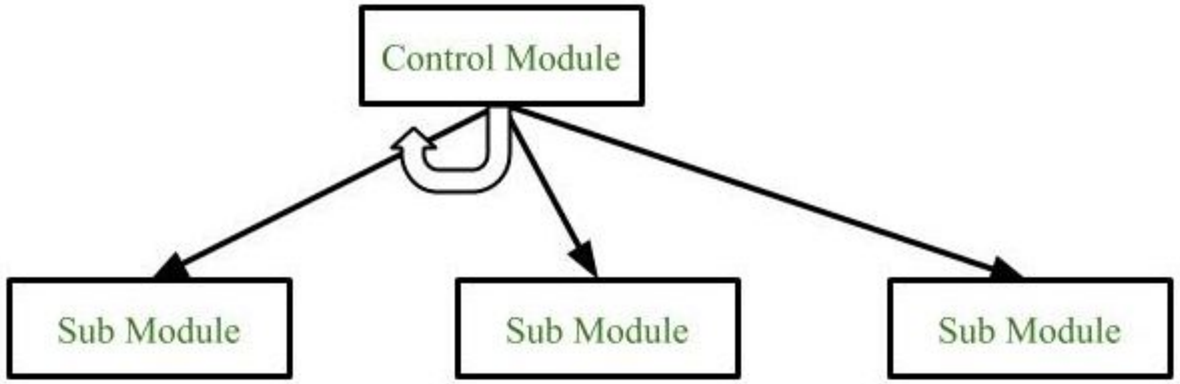
CONDITIONAL CALL

- It represents that control module can select any of the sub module on the basis of some condition.



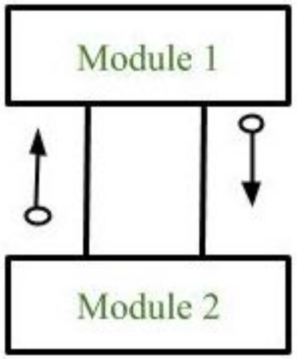
LOOP (REPETITIVE CALL OF MODULE)

- It represents the repetitive execution of module by the sub module.
- A curved arrow represents loop in the module.



DATA FLOW

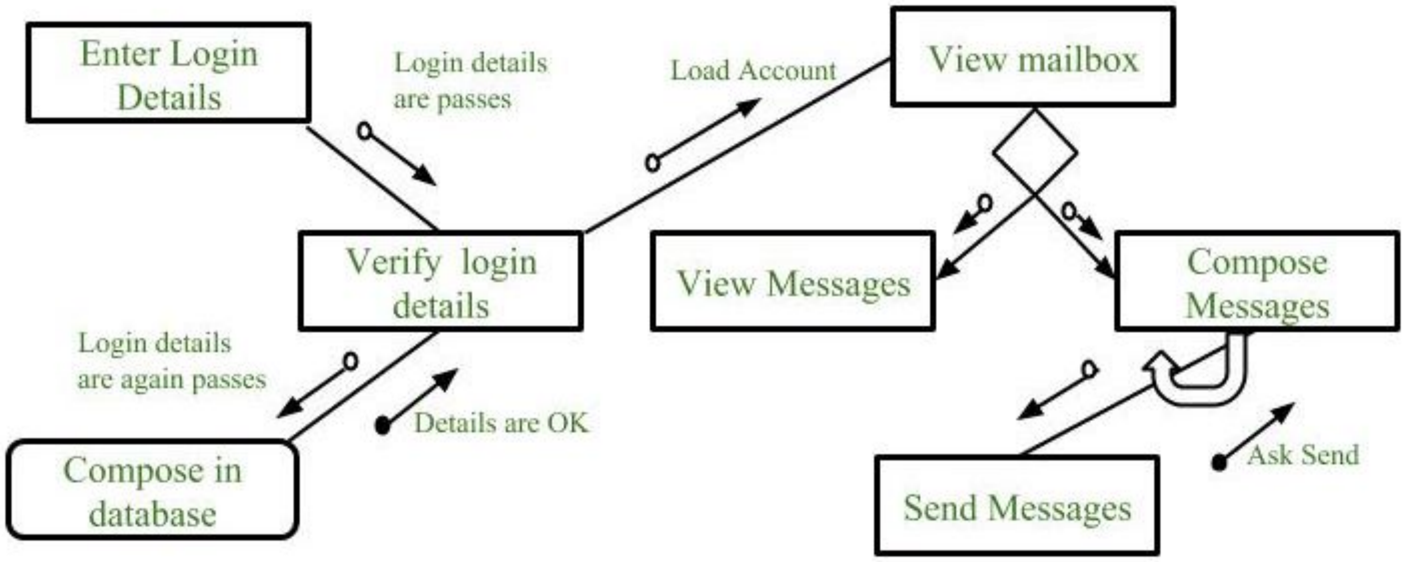
- It represents the flow of data between the modules. It is represented by directed arrow with empty circle at the end.



PHYSICAL STORAGE

- Physical Storage is that where all the information are to be stored.

STRUCTURE CHART FOR AN EMAIL SERVER



PSEUDO CODE

- It is a methodology that allows the programmer to represent the implementation of an algorithm.
- cooked up representation of an algorithm.
- These include while, do, for, if, switch. Examples below will illustrate this notion

PSEUDO CODE EXAMPLE 1

```
If student's grade is greater than or equal to 60
        Print "passed"
else
        Print "failed"
```

PSEUDO CODE EXAMPLE 2

Pseudocode

```
While the power of 2 is not larger than 1000
keep on getting the values of the power of 2.
```

C++ statement

```
int product = 2;
while ( product <= 1000 )
product = 2 * product;
```

REFERENCES

1. Software Engineering: A practitioner's Approach, Roger S Pressman, Sixth Edition. McGraw-Hill International Edition, 2005.
2. Software Engineering: Ian Summerville, Seventh Edition, Pearson Education, 2004.
3. Daniel Galin, "Software Quality Assurance", Pearson Publication, 2009.

Thank You