UNIT-1
Introduction

1.Introduction to Software engineering

1.What is Software?

“ Software is a set of instructions to acquire inputs and to manipulate them to produce the desired output in terms of functions and performance as determined by the user of the software. It also include a set of documents, such as the software manual , meant for users to understand the software system.”

1.Description of the Software

A software is described by its capabilities. The capabilities relate to the functions it executes, the features it provides and the facilities it offers. Software written for Sales-order processing would have different functions to process different types of sales order from different market segments . The features for example , would be to handle multi-currency computing, updating product , sales and Tax status. The facilities could be printing of sales orders, email to customers and reports to the store department to dispatch the goods.

1.Classes of Software

Software is classified into two classes:

- **Generic Software**:

  is designed for broad customer market whose requirements are very common, fairly stable and well understood by the software engineer.

- **Customized Software:**

  is developed for a customer where domain , environment and requirements are being unique to that customer and cannot be satisfied by generic products.

1.What is Good Software?

Software has number of attributes which decide whether it is a good or bad . The definition of a good software changes with the person who evaluates it. The software is required by the customer , used by the end users of an organization and developed by software engineer . Each one will evaluate the different attributes differently in order to decide whether the software is good.

1.What are the attributes of good software?

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

- **Maintainability**
  - Software must evolve to meet changing needs
- **Dependability**
  - Software must be trustworthy
- **Efficiency**
  - Software should not make wasteful use of system resources
- **Usability**
  - Software must be usable by the users for which it was designed

1.Software - Characteristics

- Software has a dual role. It is a product, but also a vehicle for delivering a product.
- Software is a logical rather than a physical system element.
- Software has characteristics that differ considerably from those of hardware.
- - Software is developed or engineered, it is not manufactured in the classical sense.
- - Software doesn’t “wear out”.
- - Most software is custom-built, rather than being assembled from existing components.

1.Types of Software

- **System Software-** A collection of programs written to service other programs at system level.

  For example, compiler, operating systems.

- **Real-time Software-** Programs that monitor/analyze/control real world events as they occur.

- **Business Software-** Programs that access, analyze and process business information.

- **Engineering and Scientific Software -** Software using “number crunching” algorithms for different science and applications. System simulation, computer-aided design.

1.Types of Software

- **Embedded Software-:**

  Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. It has very limited and esoteric functions and control capability.

- **Artificial Intelligence (AI) Software**:

  Programs make use of AI techniques and methods to solve complex problems. Active areas are expert systems, pattern recognition, games

1.Types of Software

- **Internet Software :**

  Programs that support internet accesses and applications. For example, search engine, browser, e-commerce software, authoring tools.

- **Software Tools and CASE environment :**

  Tools and programs that help the construction of application software and systems. For example, test tools, version control tools.

1.Software Engineering

- “A systematic approach to the analysis, design, implementation and maintenance of software.”

  (*The Free On-Line Dictionary of Computing)*
- “ The systematic application of tools and techniques in the development of computer-based applications.”

  (Sue Conger in *The New Software Engineering)*
- “ Software Engineering is about designing and developing high-quality software.”

  (Shari Lawrence Pfleeger in *Software Engineering -- The Production of Quality Software)*

1.What is Software Engineering?

Although hundreds of authors have developed personal definitions of software engineering, a definition proposed by Fritz Bauer[NAU69] provides a basis:

- **"[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."**

The IEEE [IEE93] has developed a more comprehensive definition when it states:

- **"Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)."**

1.What is Software Engineering?

- *Pressman's view:*
  "Software engineering is a layered technology (Figure 2.1)"

1.What is Software Engineering?

- **Software methods:**
- Software engineering methods provide the technical “how to” for building
- software. Methods --> how to encompass a broad array of tasks:

  - requirements analysis, design, coding, testing, and maintenance
- Software engineering methods rely on a set of basic principles.

1.What is Software Engineering?

- **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.

  It forms a basis for:

  - project management, budget and schedule control
  - applications of technical methods
  - product quality control

1.What is Software Engineering?

- **Software tools:**

  - programs provide automated or semi-automated support for the process and methods.

  - programs support engineers to perform their tasks in a systematic and/or automatic manner.

1.Why Software Engineering?

- **Objectives:**

- 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.

1.Why 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.

2.Components Of Software Engineering

2.Components of Software Engineering

SE approach has two components , namely systems engineering approach and development engineering approach. The software and its quality depends upon the system in which it is installed.

The system here has a broad meanings. The understanding of the system can be achieved by the System study and Analysis.

the System study and Analysis is carried out through SEM(Systems Engineering and Methodology). The SEM steps are as under:

- Define the Objective of the system
- Define the boundaries of the system

2.Components of Software Engineering

- Factories the system into different components
- Understand the relationship between various components
- Define relationship in terms of inputs, outputs and processes
- Understand the role of hardware and software
- Identify the key operational and functional requirements
- Model the system for analysis and development
- Discuss the system with the customer

2. Components of Software Engineering

Development Engineering methodology has of translating the system requirements as software system goal , and proceeds to achieve it through a series of steps. The development engineering steps are

- Requirement definition and specification
- Design solution to deliver the requirements
- Determine the architecture for the delivery of solution
- Customer development and planning
- Software testing components
- Integration of system components
- Implementation

2.Components of Software Engineering

Software development engineering is carried out in two ways

- Structured System Analysis and Design (SSAD)
- Object Oriented System Analysis and Design (OOSAD)

Structured System Analysis and Design (SSAD)

The SSAD approach in which the system and its requirements are decomposed in structured manner. Software development is carried out using sub-system structure, tested and integrated and implemented.

2.Components of Software Engineering

- **Object Oriented System Analysis and Design (OOSAD)**

  In contrast , the OOSAD development approach recommended the analysis of domain and builds objects of model independent of the system under consideration.

  The object could represents a function , process or document evolved for the organization. Each object has attributes that describes the methods to perform and relationship to other objects.

3.Software Crisis

3.Software Crisis

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

3.Software Crisis

- 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

3.Software Crisis

- Software product size is increasing exponentially
  - faster, smaller, cheaper hardware
- Software is everywhere: from TV sets to cell-phones
- Software is in safety-critical systems
  - cars, airplanes, nuclear-power plants
- We are seeing more of
  - distributed systems
  - embedded systems
  - real-time systems
    - These kinds of systems are harder to build
- Software requirements change
  - software evolves rather than being built

3. Software Crisis

- Software's chronic crisis: Development of large software systems is a challenging task
  - 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

- Software engineering focuses on addressing challenges that arise in development of large software systems using a systematic, disciplined, quantifiable approach

3.No Silver Bullet

- In 1987, in an article titled:

  “No Silver Bullet: Essence and Accidents of Software Engineering”

  Frederick P. Brooks made the argument that there is no silver bullet that can kill the werewolf software projects

- Following Brooks, let’s philosophize about software a little bit

3.Inherent Difficulties in Software

- Software has the following properties in its *essence*:
  - Complexity
  - Conformity
  - Changeability
  - Invisibility
- Since these properties are not *accidental* representing software in different forms do not effect them

4.Software Engineering Processes

4.What is Software Engineering Process?

- **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.

  It forms a basis for:

  - project management, budget and schedule control
  - applications of technical methods
  - product quality control

4.Why Software Engineering Process?

- 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.

4.Why Software Engineering Process?

- **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.

5.Similarities And Differences From Conventional Engineering Processes

5.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.

  3. A pastime similar to banging one's head against a wall, but with fewer opportunities for rewards. (2 and 3 from *The New Hacker's Dictionary)*

- **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."

5.What is the difference between software engineering and computer science?

Computer Science is concerned with **Software Engineering**

- theory
- fundamentals

- the practicalities of developing
- delivering useful software

omputer 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

5.What is the 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**

6.Software Quality Attributes

6.Software quality

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

6.McCall’s quality factors

- Two categories of software quality factors:
  - Factors that can be directly measured (bugs or defects)
  - Factors that can be measured only indirectly (usability or maintainability)
- Both sets can (and should) be measured

6.McCall's quality factors

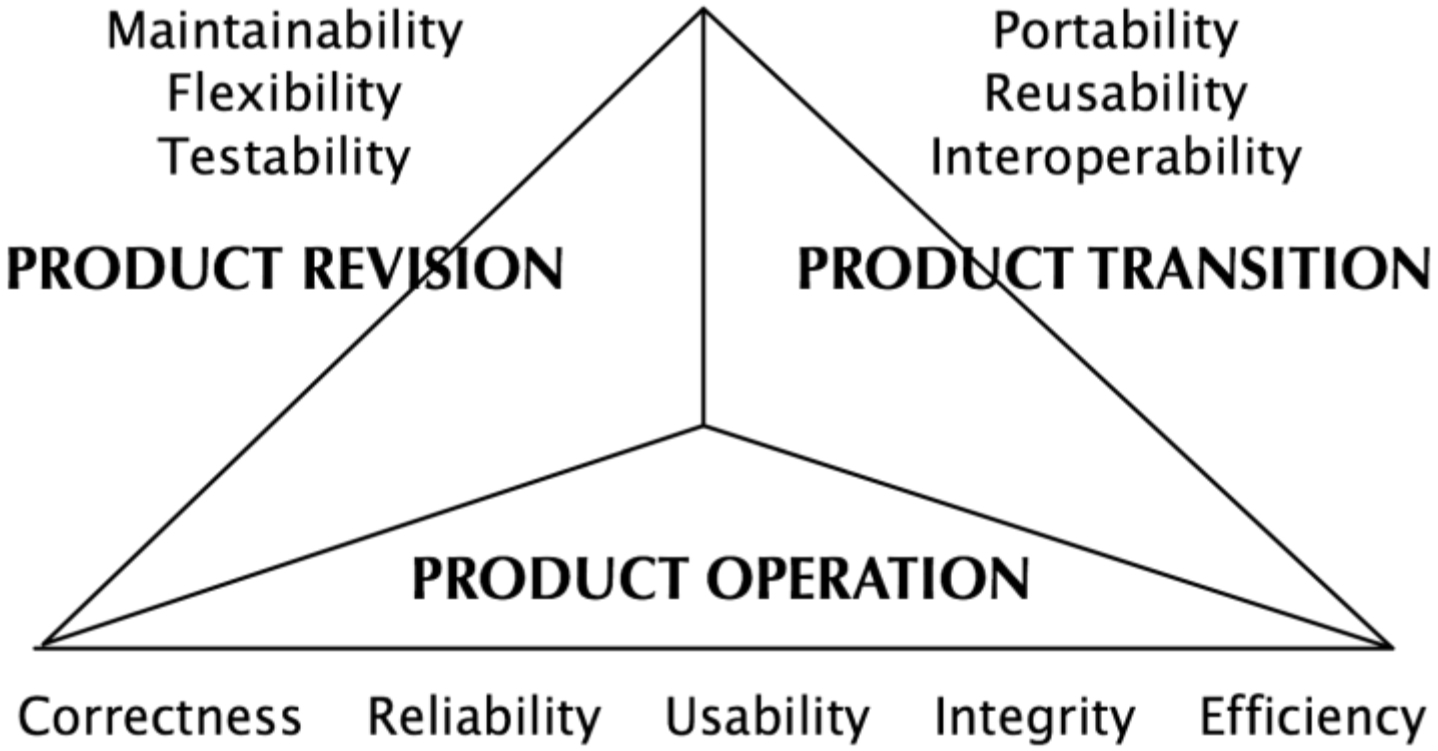
- McCall, Richards, and Walters group these factors into three categories, focused on three aspects of a software product:
  - Its operational characteristics

    (PRODUCT OPERATION)
  - Its ability to undergo change

    (PRODUCT REVISION)
  - Its adaptability to new environments

    (PRODUCT TRANSITION)

6.McCall’s quality factors



6.Product Operation

- Correctness:
  - the extent to which a program satisfies its specifications and fulfils the customer's mission objectives
- Reliability:
  - the extent to which a program can be expected to perform its intended function with required precision
- Usability:
  - the amount of computing resources and code required by the program to perform its function

6.Product Operation (cont.)

- Integrity:
  - the extent to which access to software or data by unauthorized persons can be controlled
- Efficiency:
  - the effort required to learn, operate, prepare input for, and interpret output of a program

6.Product Revision

- Maintainability:
  - the effort required to locate and fix an error in a program
- Flexibility:
  - the effort required to modify an operational program
- Testability:
  - the effort required to test a program to ensure that it performs its intended function

6.Product Transition

- Portability:
  - the effort required to transfer the program from one hardware and/or software system to another
- Reusability:
  - the extent to which a program (or parts or a program) can be reused in other applications
- Interoperability:
  - the effort required to couple one system to another

6.McCall’s quality factors

- It is difficult (and sometimes impossible) to directly measure all of these quality factors
- Many of McCall’s metrics can only be measured subjectively
- The metrics may be in the form of a checklist to “grade” attributes of the software

7. SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

7.Software Life Cycle Model

- From initial development to release, running and maintenance, the software

  life span is divided into different phases.

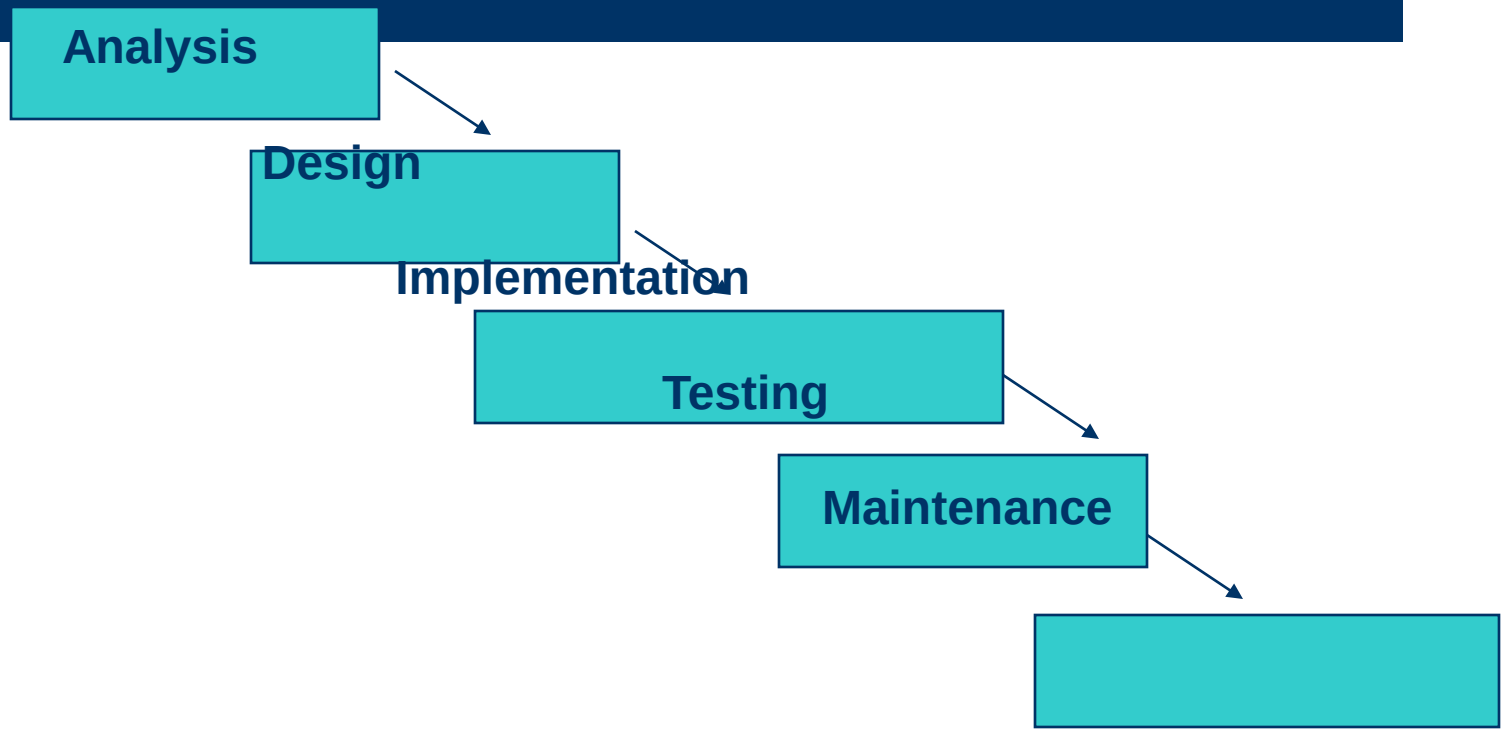
- Each phase has a milestone product.

7. Software Life Cycle Model

Different life cycle models reflects different software development methodologies

– Waterfall Model
– Prototyping Model
– Spiral Model
_Iterative Enhancement Model
_Evolutionary Development Model

The Waterfall Life Cycle Model



8.WATERFALL MODEL

8.The Waterfall Life Cycle Model

The earliest life cycle model

- Completion of one phase is identified by the attainment of a set of associated deliverables.

- The deliverables form the contract for the next phase.

8. The Waterfall Life Cycle Model

- This approach imposes a linear [redacted] documents from successive phases with no opportunity to revise previous phases.

- Handles requirement changes badly, so, customer need have a clear idea of their requirements.

- Design decisions are difficult to reverse.

- Long delay before a program is delivered.

8. CLASSICAL WATERFALL MODEL

The classical waterfall model divides the life cycle into the phases shown in below figure. This model breaks down [redacted]ifferent phases of this model are: feasibility study, requirements analysis and specification, design, coding and unit testing, integration and system testing, and maintenance. The different phases starting from the feasibility study to the integration and system testing phase are known as the development phases. The part of the life cycle model between the feasibility study and product testing and delivery is known as the development part. At the end of the development part of the life cycle, the product becomes ready to be delivered to the customer. The maintenance phase commences after completion of the development phase.

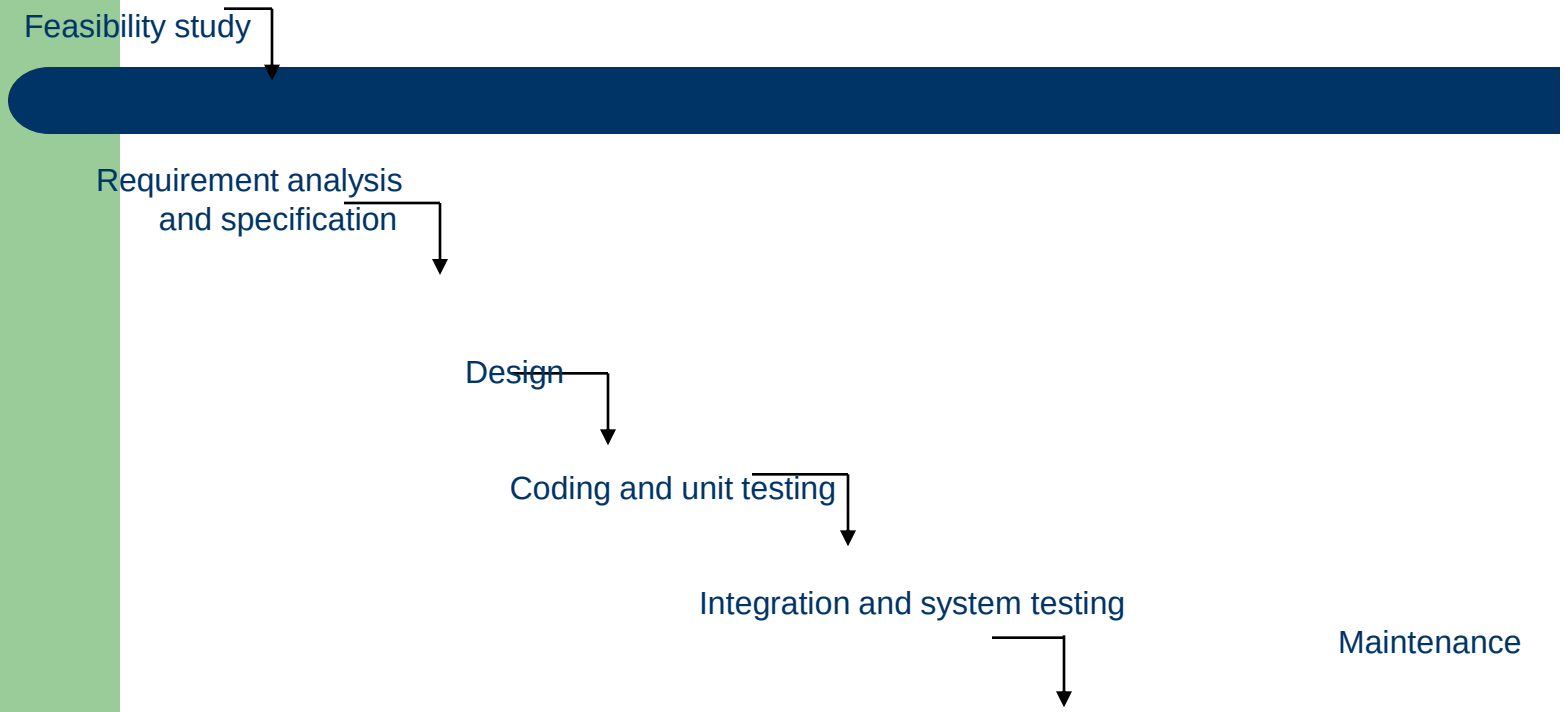


Figure 2.1: Classical waterfall model.

8. Feasibility Study

- The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system. The collected data are analyzed to arrive at the following:
- An abstract problem definition. An abstract problem definition is a rough description of the problem which considers only the important requirements and ignores the rest.
- Formulation of the different solution strategies.
- Analysis of alternative solution strategies to compare the benefits and shortcomings.

2.1.2 Requirements Analysis and Specification

- The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely requirements gathering and analysis, and requirements specification.

8. Requirements gathering and analysis

- The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed with a view to clearly understanding the customer requirements and weeding out the incompleteness and inconsistencies in these requirements

- The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions.

- For example, to perform the requirements analysis of a business accounting software required by on organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore, it is necessary to identify all ambiguities and contradictions in the requirements and resolved them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

8. Requirements specification

- The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document.
- The important components of this document are the functional requirements, the nonfunctional requirements, and the goal of implementation.
- Documenting the functional requirements involves the identification of the functions to be supported by the system.
- [redacted]red on the input data, and the output data to be produced.
- The non-functional requirements identify the performance requirements, the required standards to be followed, etc.
- The SRS document is written using the end –user terminology. This makes the SRS document understandable by the customer. After all, it is important that the SRS document be reviewed and approved by the customer.
- The SRS document normally serves as a contract between the development team and the customer.
- Any future dispute between the customer and the developers can be settled by examining the SRS document.
- It is therefore an important document which must be thoroughly understood by the developer team, and reviewed jointly with the customer.
- The SRS documents provides not only the basis for carrying out all the development activities, but also the basis for several other documents such as the design document, the users' manuals, the system test plan, etc.

8. Design

- The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the soft ware architecture is derived from the SRS document. Two distinctly different design approaches are available: the traditional design approach

Traditional design approach

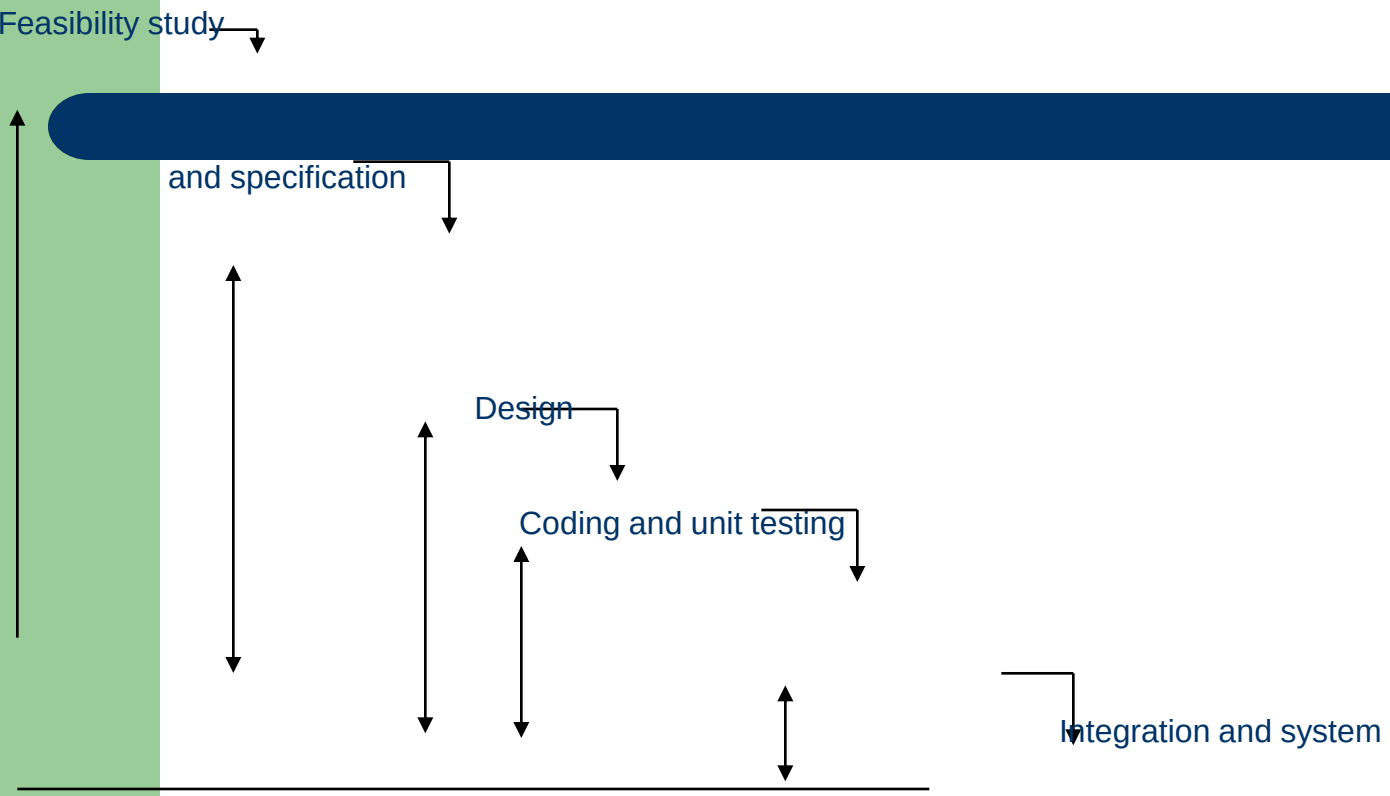
- The traditional design approach is currently being used by many software development houses. Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structure design activity. During structured design, the results of structured analysis are transformed into the software design.
- Structured analysis involves preparing a detailed analysis of the different functions to be supported by the system and identification of the data flow among the different functions. Data flow diagrams (DFDs) are used to perform structured analysis and to document the results.

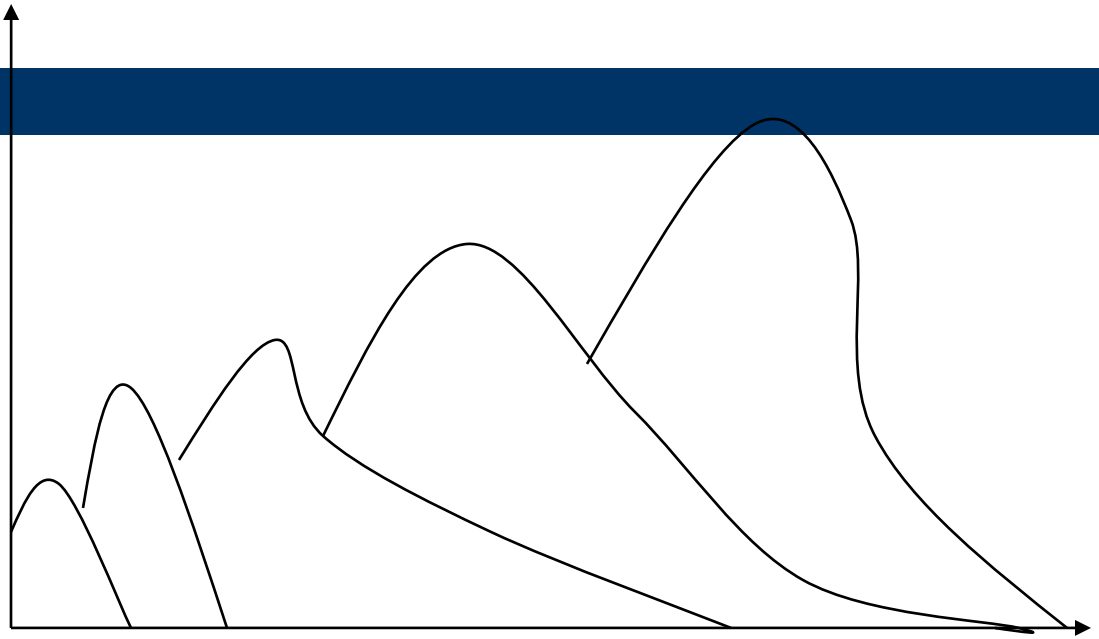
- **Object-oriented design approach**
- Object-oriented design (OOD) is a relatively new technique. In this technique, various objects that occur in the problem domain and the solution domain are first identified and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design. The OOD approach has several benefits such as lower development time and effort, and better maintainability of the product.
- **Coding and Unit Testing**
- The purpose of the coding and unit testing phase of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. A coding standard addresses issues such as the standard ways of laying out of the program codes, the template for laying out the function and module headers, commenting guidelines, variable and function naming conventions, the maximum number of source lines permitted in each module, and so forth.
- During the phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

- **Integration and System Testing**
- During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out.
- Alpha-testing: It is the system testing performed by the development team,
- Beta-testing: It is the system testing performed by a friendly set of customers.
- Acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.
- **Maintenance**
- Maintenance of a typical software product requires much more effort than the effort necessary to develop the product itself. Maintenance involves performing any one or more of the following three kinds of activities:
- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.
-

- **8. ITERATIVE WATERFALL MODEL**

- Feedback paths are needed in the classical waterfall model from every phase to its preceding phases as shown in figure to allow for the correction of the errors committed during a phase that are detected in later phases.
- Some of the glaring shortcomings of the waterfall model are the following:
- The waterfall model cannot satisfactorily handle the different types of risks that a real-life software project is subjected to. For example, the waterfall [redacted] ore the rest of the development activity can start. Therefore, it cannot accommodate the uncertainties that exist at the beginning of most of the projects. As a result, it cannot be satisfactorily used in projects where only rough requirements are available at the beginning of the project.
- To achieve better efficiency and higher productivity, most real-life projects cannot follow the rigid phase sequence imposed by the waterfall model. A rigid adherence to the waterfall model creates “blocking states” in the system. That is, some team members would have to wait for a phase to be complete before they can start their next activity. This is clearly a wastage of resources and such wastages are rarely tolerated in real projects. For example, if an engineer works fast and completes the design of the parts assigned to him early, he should not idle, waiting for the others to complete their designs. Instead, he should proceed to the next phase.


Feasibility study
and specification
Design
Coding and unit testing
Integration and system

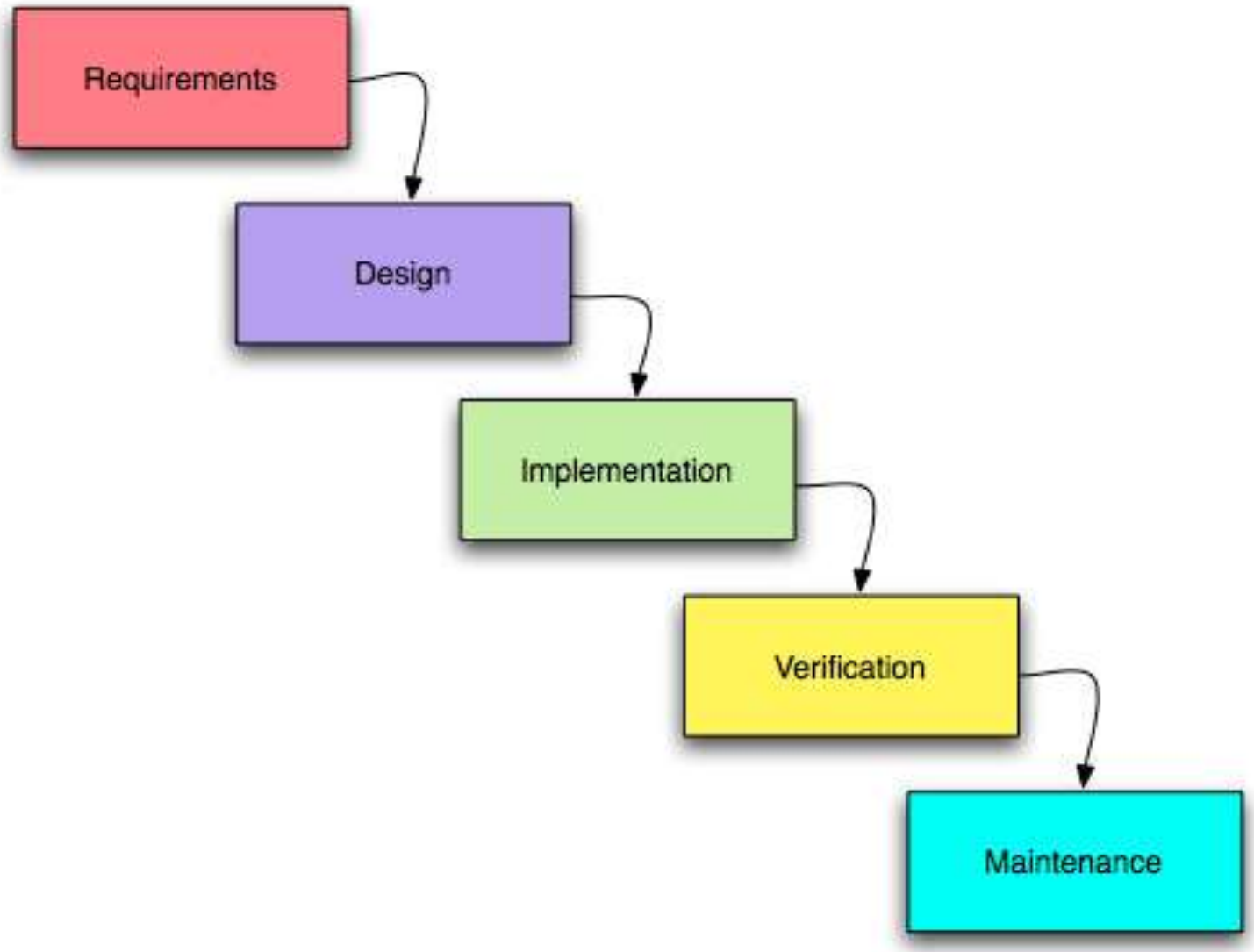


Distribution of effort over time for various phases in the iterative waterfall model.

- Some of the glaring shortcomings of the waterfall model are the following:
- The waterfall model cannot satisfactorily handle the different types of risks that a real-life software project is subjected to. For example, the waterfall model assumes that the requirements be completely specified before the rest of the development activity can start. The [redacted] at exist at the beginning of most of the projects. As a result, it cannot be satisfactorily used in projects where only rough requirements are available at the beginning of the project.
- To achieve better efficiency and higher productivity, most real-life projects cannot follow the rigid phase sequence imposed by the waterfall model. A rigid adherence to the waterfall model creates “blocking states” in the system. That is, some team members would have to wait for a phase to be complete before they can start their next activity. This is clearly a wastage of resources and such wastages are rarely tolerated in real projects. For example, if an engineer works fast and completes the design of the parts assigned to him early, he should not idle, waiting for the others to complete their designs. Instead, he should proceed to the next phase.

- WATERFALL MODEL(students can study this also)

Saturday, August 29, 12:04:10 PM



Saturday, August 29, 12:04:10 PM

The Waterfall Model consists of the following steps:

- **System Conceptualization.**

System Conceptualization refers to the consideration of all aspects of the targeted business function or process, with the goals of determining how each of those aspects relates with one another, and which aspects will be incorporated into the system.

Saturday, August 29, 12:04:10 PM

- **Systems Analysis.** This is the second phase refers to the gathering of system requirements, with the goal of determining how these requirements will be accommodated in the system. Extensive communication between the customer and the developer is essential.

Saturday, August 29, 12:04:10 PM

- **System Design.** Once the requirements have been collected and analyzed, it is necessary to identify in detail how the system will be constructed to perform necessary tasks. More specifically, the System Design phase is focused on the data requirements
- (what information will be processed in the system?),
- the software construction (how will the application be constructed?),
- and the interface construction (what will the system look like?
- What standards will be followed?).

Saturday, August 29, 12:04:10 PM

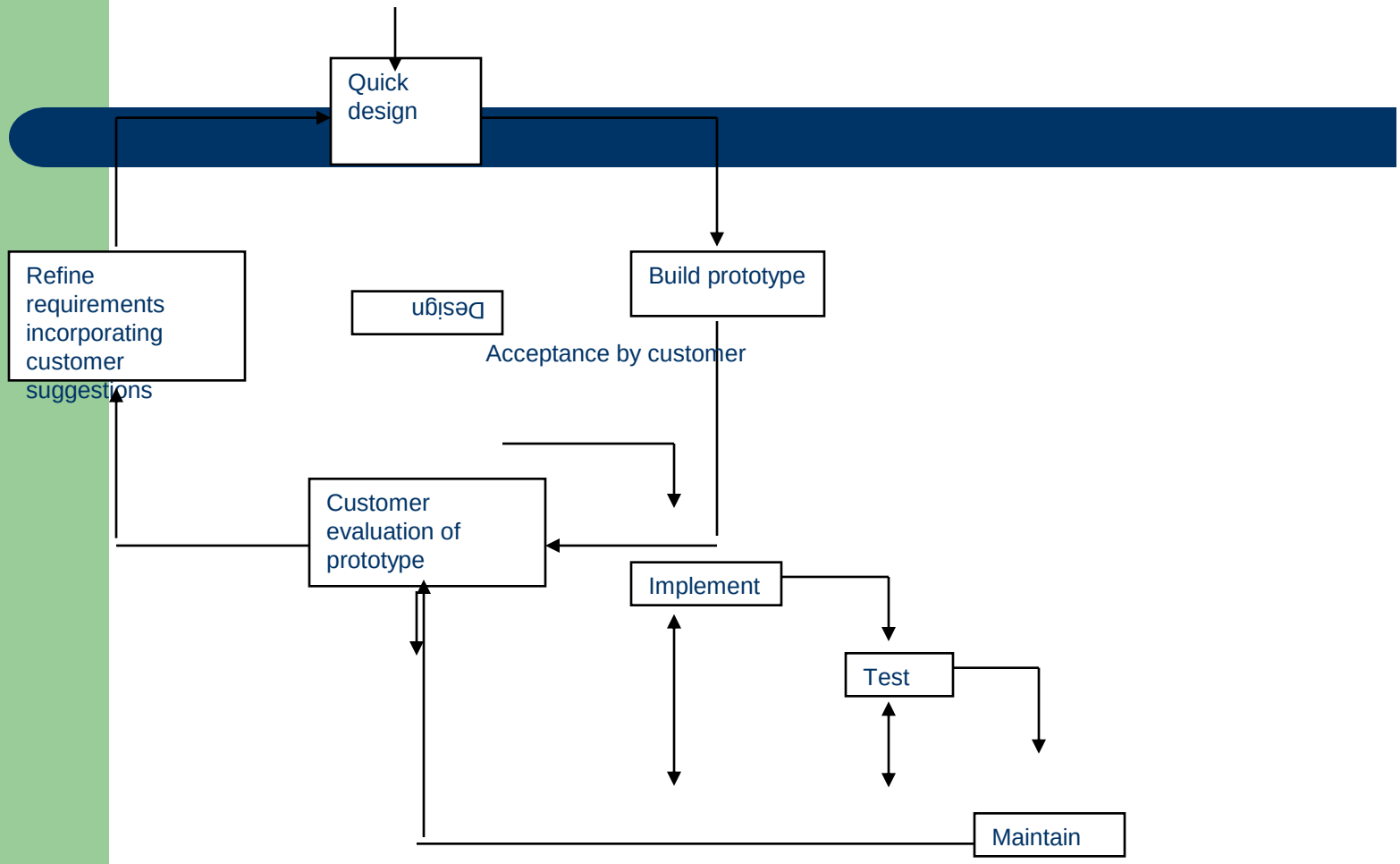
- **Coding.** Also known as programming or the coding phase, this step involves the creation of the system software. Requirements and systems specifications from the System Design step are translated into machine-readable computer code.

Saturday, August 29, 12:04:10 PM

- **Testing.** As the software is created and added to the developing system, testing is performed to ensure that it is working correctly and efficiently.
- internal efficiency -The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented.
- external effectiveness- The goal of external effectiveness testing is to verify that the software is functioning according to system design, and that it is performing all necessary functions or sub-functions. Testing can be a labor-intensive process, due to its iterative.

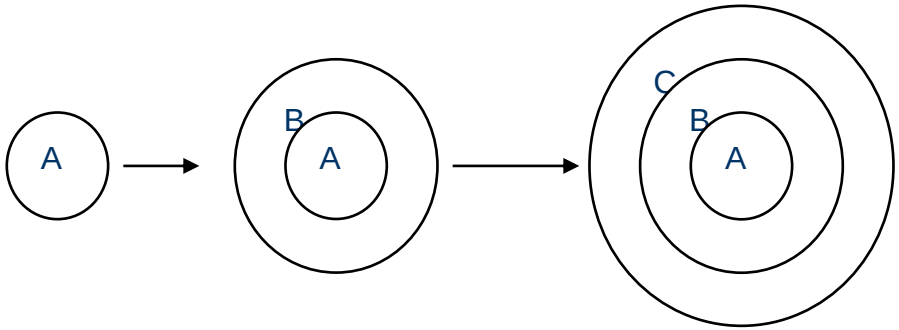
9. PROTOTYPING MODEL

- **9. PROTOTYPING MODEL**
- The prototyping model suggests 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. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. An important purpose is to illustrate the input data [redacted] customer. This is a valuable mechanism for gaining better understanding of the customer's needs. In this regard, the prototype model is very useful in developing the Graphical User Interface (GUI) part of a system.
- The prototyping model of software development is shown in figure. In this model, product development starts with an initial requirements gathering phase. A quick design is carried out and the prototype is built. The developed prototype is submitted to the customer for his evaluation. Based on the customer feedback, the requirements are refined and the prototype is suitably modified. This cycle of obtaining customer feedback and modifying the prototype continues till the customer approves the prototype. The actual system is developed using the iterative waterfall approach. However, in the prototyping model of development, the requirements analysis and specification phase becomes redundant as the working prototype approved by the customer becomes redundant as the working prototype approved by the customer becomes an animated requirements specification.



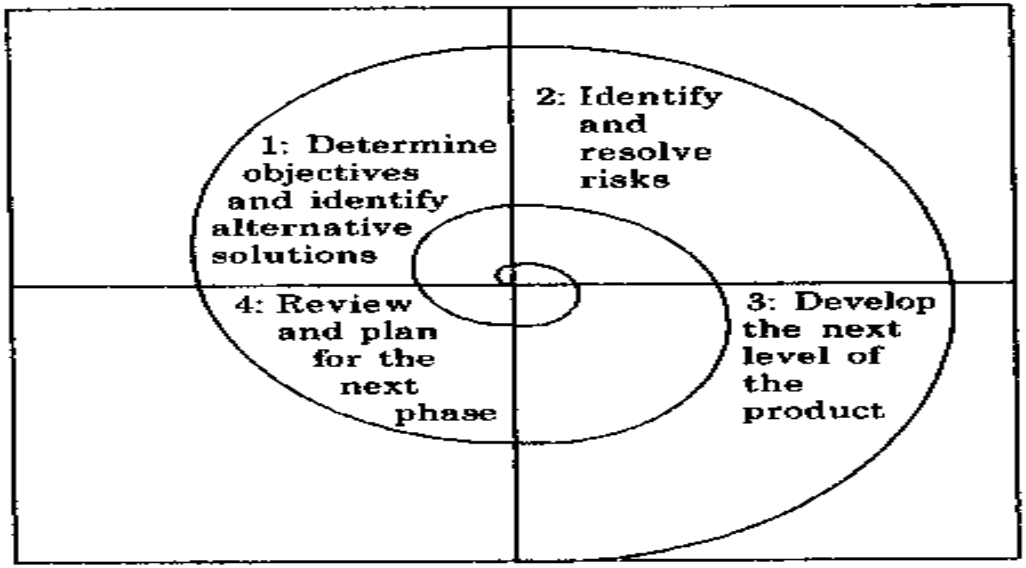
Prototyping model of software development

10. SPIRAL MODEL


A
B
A
C
B
A

10. SPIRAL MODEL

The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. This model is much more flexible compared to the other models, since the exact number of phases through which the product is developed in this model is not fixed. The phases we mentioned are just examples and may vary from one project to another. For instance, it is possible that for some other project the design might be accomplished over three or four consecutive loops and in some other project the design might be over in just one loop.



Spiral model of software development

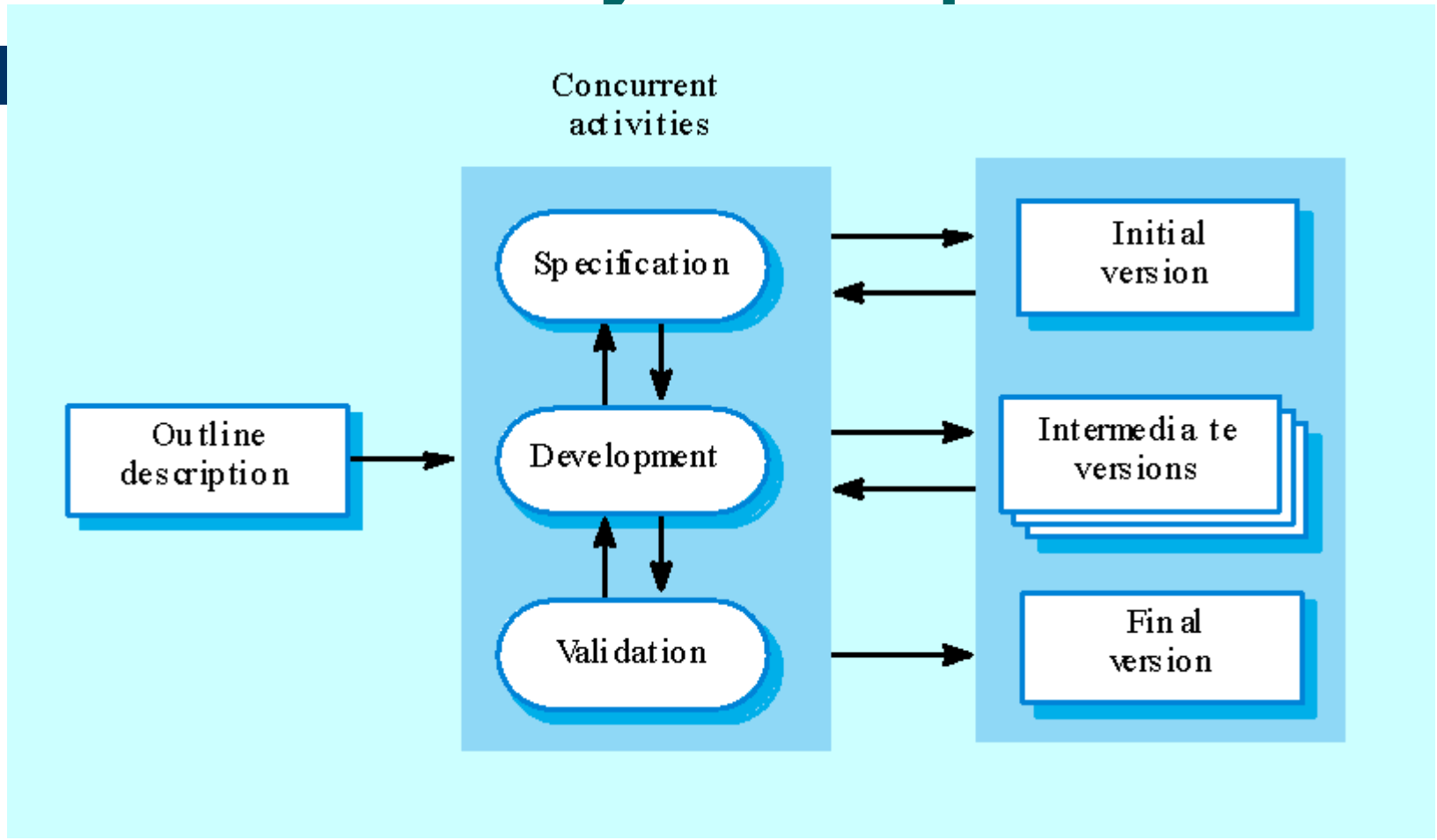
- Each phase in this model is split into four sectors as shown in figure. The first quadrant identifies the objectives of the phase and the alternative solutions possible for the phase under consideration. During the second quadrant, the alternative solutions are evaluated to select the best solution possible. For the chosen solution, the potential risks are identified and dealt with by developing an [redacted] circumstance that might hamper the successful completion of a software project.
- Activities during the third quadrant consists of developing and verifying the next level of the product. Activities during the fourth quadrant concern reviewing the results of the stages traversed so far with the customer and planning the next iteration around the spiral. With each iteration around the spiral, progressively a more complete version of the software gets built. After several iterations along the spiral, all risk are resolved and the software is ready for development. At this point, a waterfall model of software development is adopted. The radius of the spiral at any point represents the cost incurred in the project till then, and the angular dimension represents the progress made in the current phase.

11. EVOLUTIONARY DEVELOPMENT MODEL

11.Evolutionary development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

11. Evolutionary development



11. Evolutionary development

- Problems
  - Lack of process visibility;
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

12. ITERATIVE ENHANCEMENT MODEL

- **Iterative approach** is a cyclic software development process developed in response to the weaknesses of the waterfall model. It starts with an **initial planning** and [redacted] between.
- The Iterative Model addresses many problems associated with the Waterfall Model.
- The basic idea behind **iterative enhancement** is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system.

- Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented.
- At each iteration, design modifications are made and new functional capabilities are added.
- In the Iterative Model **analysis** is done the same way as it is done in the Waterfall method.
- Once this **analysis** is over, each requirement is categorized based on their priority. These priorities are:
  - **High**
  - **Low**
  - **Medium**

- The Procedure itself consists of the **Initialization step, the Iteration step, and the Project Control List.**
- The **initialization step** creates a base version of the system. The [redacted]hich the user can react. It should offer a sampling of the key aspects of the problem and provide a solution that is simple enough to understand and implement easily.
- To guide the **iteration process**, a project control list is created that contains a record of all tasks that need to be performed. It includes such items as new features to be implemented and areas of redesign of the existing solution.
- The **control list** is constantly being revised as a result of the analysis phase.
- The iteration involves the redesign and implementation of a task from the project control list, and the analysis of the current version of the system.

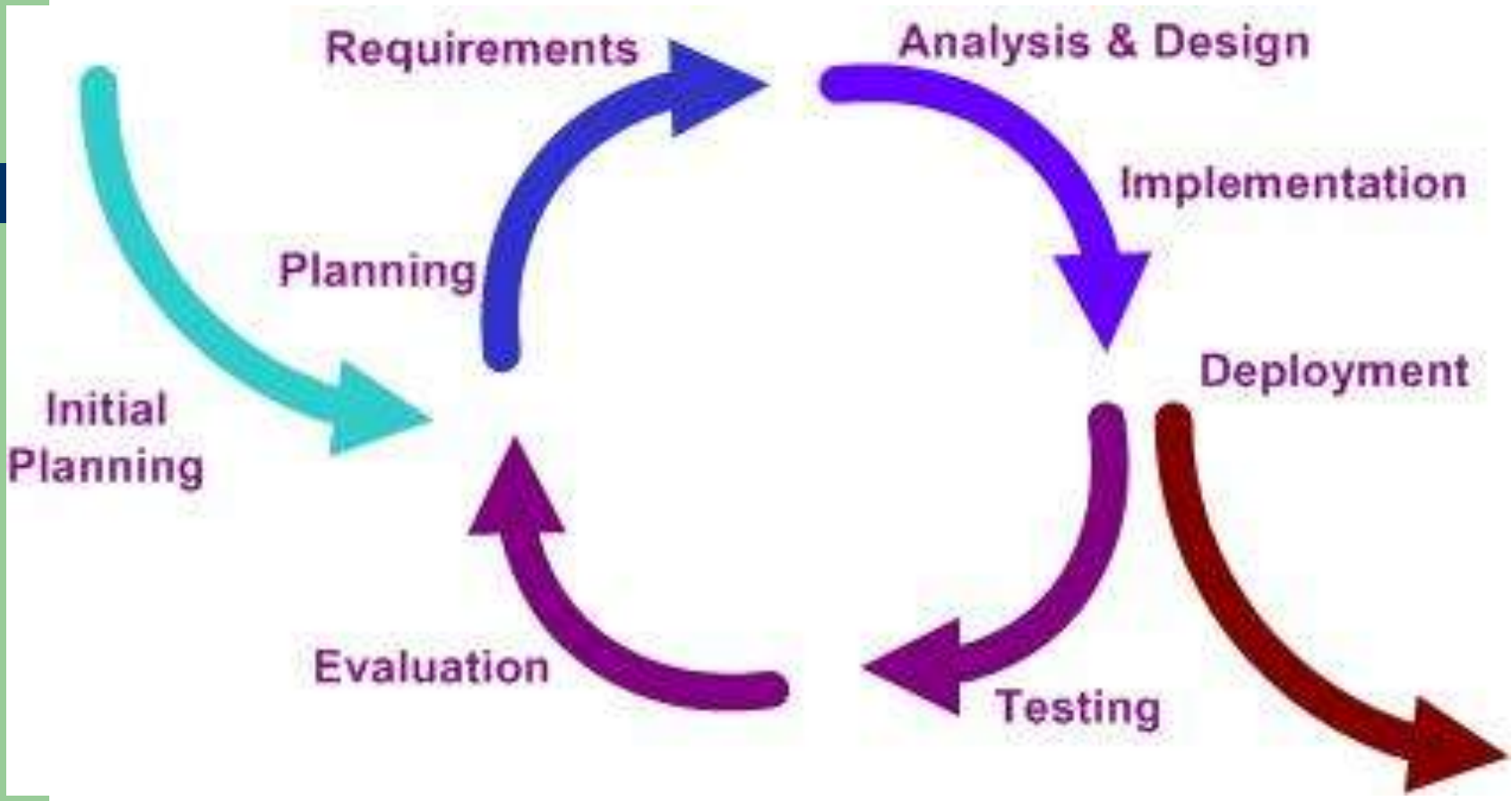


Fig.: An Iterative Model

- **Analysis.** Decide which improvement should be [redacted]
- **Design** how you will code the change.
- **Code** it and compile to make sure your coding is correct.

- **Testing.** Run the program, using test data if necessary. If it doesn't run, the program must be debugged and either the coding or design should be changed. Because the last change should have been small, it's usually easy to identify the source of the problem.

- Continue around this **loop** until the program is finished, or you've run out of resources (time). Using the iterative approach means that there's a running program at the end of each iteration. It doesn't to everything, but it is something you can turn in.

- For a company this means that it may be able to use the program for some work and get value out of it before it is finished. It also means that the project can be terminated as soon as the program is "good enough”.

Iterative Development

- The problems with the waterfall model created a demand for a new method of developing systems, which could [redacted] information, and offer greater flexibility.
- With **iterative development**, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users.
- Often, each iteration is actually a mini-waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products which are produced at the end of each step (or series of steps) can go into production immediately as incremental releases.

- Iterative development slices the deliverable business value (system functionality) into
- In each iteration a slice of functionality is delivered through cross-discipline work, starting from the model/requirements through to the testing/deployment.

Advantages of Iterative development

- You **always have a running version of the program**, e.g., if you run out **of** time, you can deliver the last iteration, which may not [redacted] worth more to your instructor than a program which doesn't even compile yet.

- It helps identify the source **of** the last error (compilation or execution), because you know it's in the code you just added in this iteration. This makes finding bugs much faster.

- It's **psychologically more satisfying** to get positive feedback on your work, i.e., a running program.

- **Corrections early in development generally take less time** than later in the development process.

- Faster Coding, testing and Design Phases
- Facilitates the support for changes within the life

PROBLEMS/CHALLENGES ASSOCIATED WITH THE ITERATIVE MODEL

- The user community needs to be actively involved throughout the project.
- While this involvement is a positive for the project, it is demanding on the time of the staff and can add project delay.
- Communication and coordination skills take center stage in project development.

- Informal requests for improvement after each phase may lead to confusion -- a controlled mechanism for handling substantive requests needs to be developed
- The iterative model can lead to "scope creep," since user feedback following each phase may lead to increased customer demands.
- As users see the system develop, they may realize the potential of other system capabilities, which would enhance their work.

- More time spent in review and analysis
- [redacted]del
- Delay in one phase can have detrimental effect on the software as a whole