Software Engineering

Organization of this Lecture:

- What is Software Engineering?
- Programs vs. Software Products
- Evolution of Software Engineering
- Notable Changes In Software Development Practices
- Introduction to Life Cycle Models
- Summary

What is Software Engineering?

- "A systematic collection of good program development practices and techniques."

- Engineering approach to develop software.

  – Software engineering discusses systematic and cost effective techniques for software development. These techniques help develop software using engineering approach.

- Systematic collection of past experience:

  – techniques,

  – methodologies,

  – guidelines.

Software + Engineering

- The term **software engineering is composed of two words, software and engineering.**

- **Software** is considered to be a collection of executable programming **code**, associated **libraries** and **documentations**. Software, when made for a specific requirement is called **software product**.

- **Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.

- So, we can define *software engineering* **as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.**

- *The outcome of software engineering is an efficient and reliable software product.*

Definition

- IEEE defines software engineering as:

  The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

Scope and necessity of software engineering

- Ex- Difference between building a Wall and a Multistoried Building.

- Without using software engineering principles it would be difficult to develop large programs.

- In industry it is usually needed to develop large programs to accommodate multiple functions.

- A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes
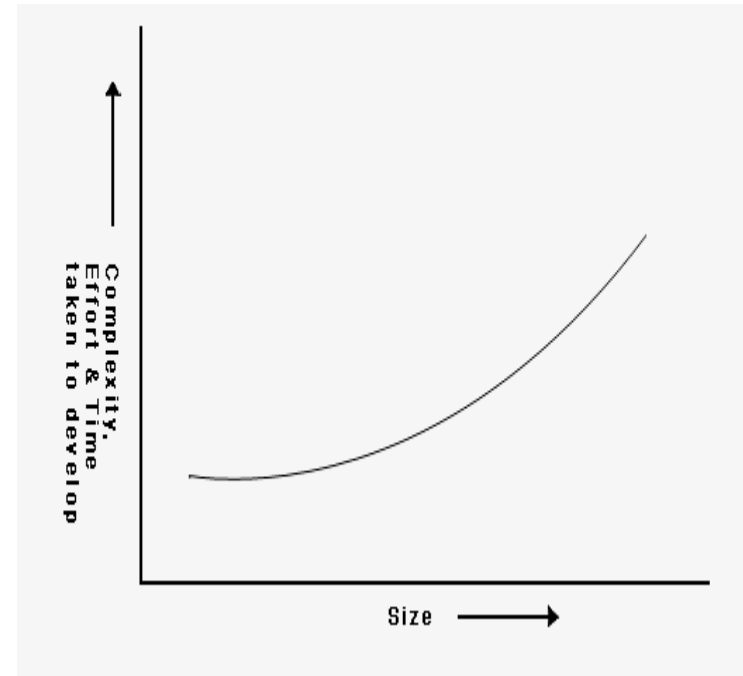
Increase in development time and effort with problem size

- For example, a program of size 1,000 lines of code has some complexity.

- But a program with 10,000 LOC is not just 10 times more difficult to develop, but may as well turn out to be 100 times more difficult **unless software engineering principles are used.**

- In such situations software engineering techniques come to rescue. Software engineering helps to reduce the programming complexity.

- Software engineering principles use two important techniques to reduce problem complexity: **abstraction and decomposition.**

- The principle of abstraction implies that a **problem can be simplified by omitting irrelevant details.**

- Consider only those **aspects of the problem that are relevant for certain purpose** and suppress other aspects that are not relevant for the given purpose.

- Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on.

- Abstraction is a powerful way of reducing the complexity of the problem.

3rd abstraction

2nd abstraction

1st abstraction

Full Problem
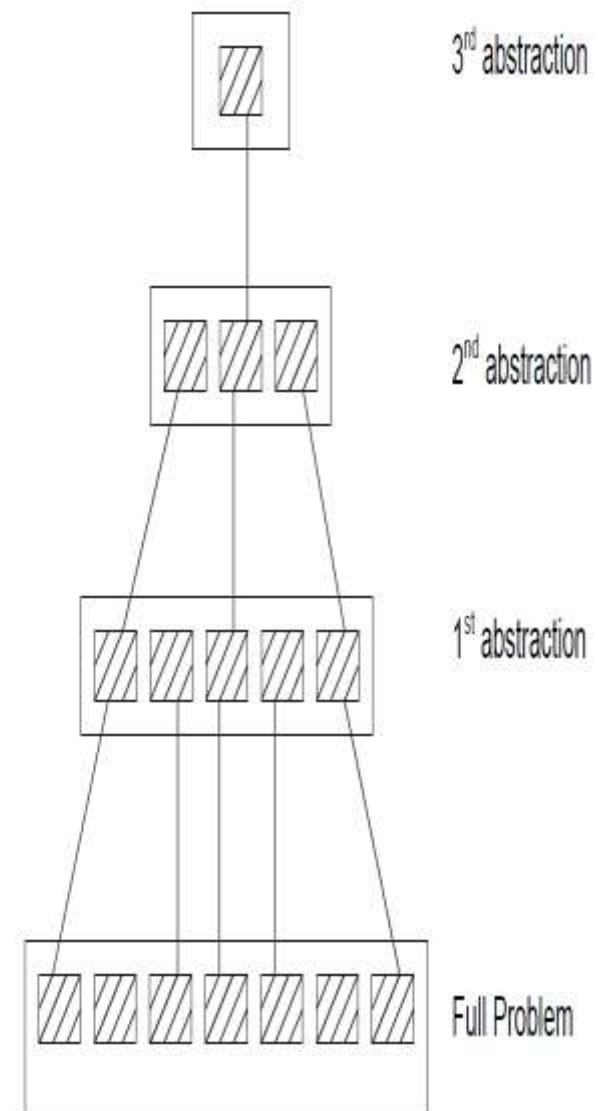
Abstraction

Problem 1 : Develop understanding on some country. How to start?

Solution

- Check maps first.

- **Political map** - It is the political abstraction, shows state, city, national boundaries. But physical aspects are ignored.

- **Physical map** - It is the physical abstraction, shows all physical features like mountains, rivers, coastlines etc. But ignores state/city etc.

Multiple abstractions are possible for the same object.In each abstraction, some properties are taken into consideration and rest are ignored.

- The other approach to tackle problem complexity is **decomposition**. A complex problem is divided into several smaller problems and then the smaller problems are solved one by one.

- However, in this technique any random decomposition of a problem into smaller parts will not help.

- The problem has to be decomposed such that each component of the **decomposed problem can be solved independently** and then the solution of the different components can be combined to get the full solution.

- **Challenge:** A good decomposition of a problem as should minimize interactions among various components.

- If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

Decomposition
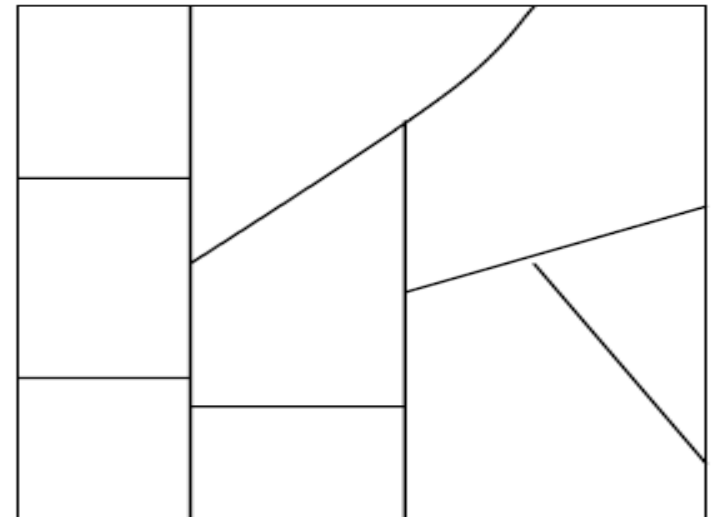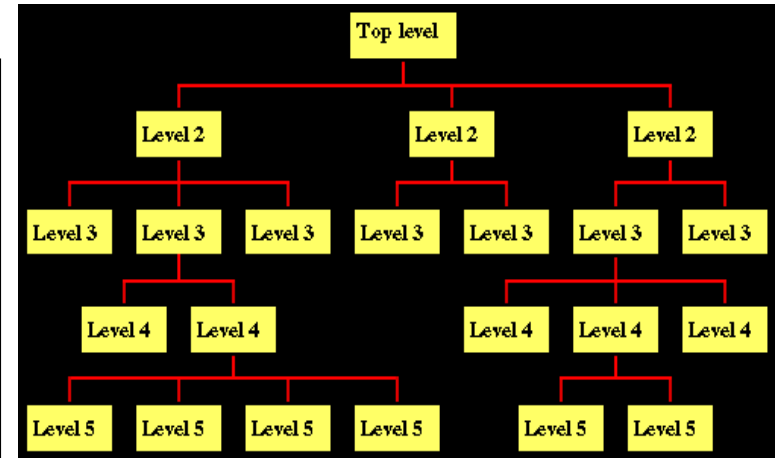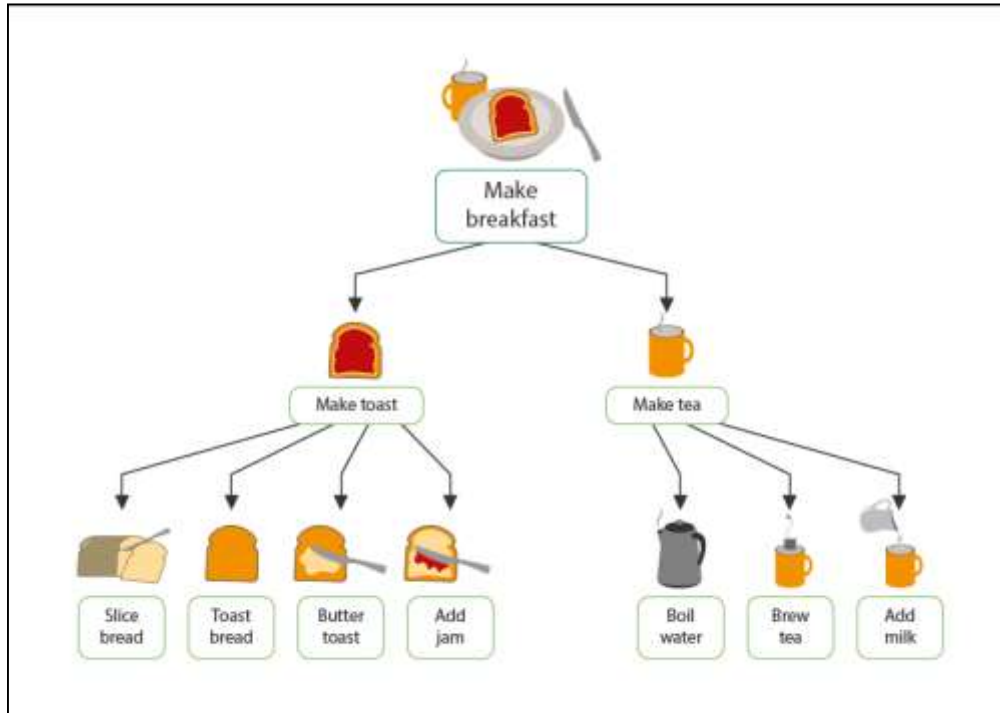
NEED OF SOFTWARE ENGINEERING

Because of higher rate of change in user requirements and environment on which the software is working.

Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

NEED OF SOFTWARE ENGINEERING

Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works.

If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

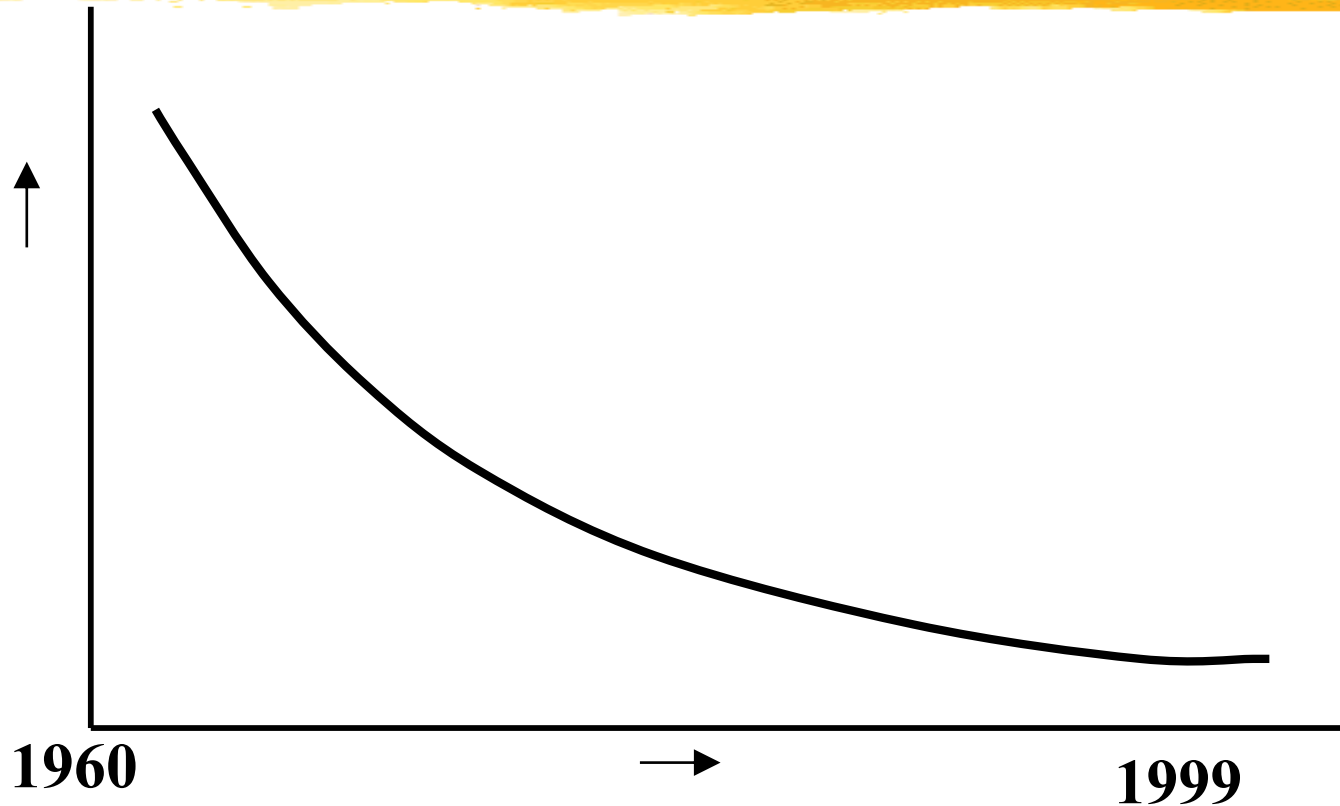**Quality Management-** Better process of software development provides better and quality software product.

Software Crisis

- Software products:
  - fail to meet user requirements.
  - frequently crash.
  - expensive.
  - difficult to alter, debug, and enhance.
  - often delivered late.
  - use resources non-optimally.

Software Crisis (cont.)

At present, organisations are spending much more on software than hardware.

Factors contributing to the software crisis

- Larger problems,
- Lack of adequate training in software engineering,
- Increasing skill shortage,
- Low productivity improvements.

Programs versus Software Products

Programs	Software Products
• Usually small in size	• Large
• Author himself is sole user	• Large number of users
• Single developer	• Team of developers
• Lacks proper user interface	• Well-designed interface
• Lacks proper documentation	• Well documented & user-manual prepared
• Ad hoc development.	• Systematic development

CHARACTERESTICS OF GOOD SOFTWARE

- A software product can be judged by what it offers and how well it can be used.

- This software must satisfy on the following grounds:
  - Operational
  - Transitional
  - Maintenance

Operational

- This tells us how well software works in operations. It can be measured on:
  - Budget : in terms of cost, manpower
  - Usability : ease of use
  - Efficiency : minimum expenditure of time and effort
  - Correctness: with respect to a specification
  - Functionality : having a practical use
  - Dependability : extent to which a critical system is trusted by its users
  - Security : degree of resistance or protection from

Transitional

- This aspect is important when the software is moved from one platform to another:

    - Portability : usability of the same software in different environments

    - Interoperability : ability of a system or a product to work with other systems

    - Reusability : use of existing assets in some form within the software product development process

    - Adaptability : able to change or be changed in order to fit or work better in some situation

- *C compilers are readily available for the majority of operating systems, which in turn makes C programs very portable*

- *Interoperability is the design of things to work together- a contact management app might integrate with a sales force automation tool to sync sales contacts.*

- *The software library is a good example of code reuse.*

- *Adaptability - a)dual cell phone changes switches sim. b)if user presses character keys while entering numeric data the software can be designed to ignore such incorrect key presses.*

Maintenance

- This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

  – Modularity: degree to which a system's components may be separated and recombined

  – Maintainability: degree to which an application is understood, repaired, or enhanced

  – Flexibility: ability for the solution to adapt to possible or future changes in its requirements

  – Scalability: ability of a program to scale

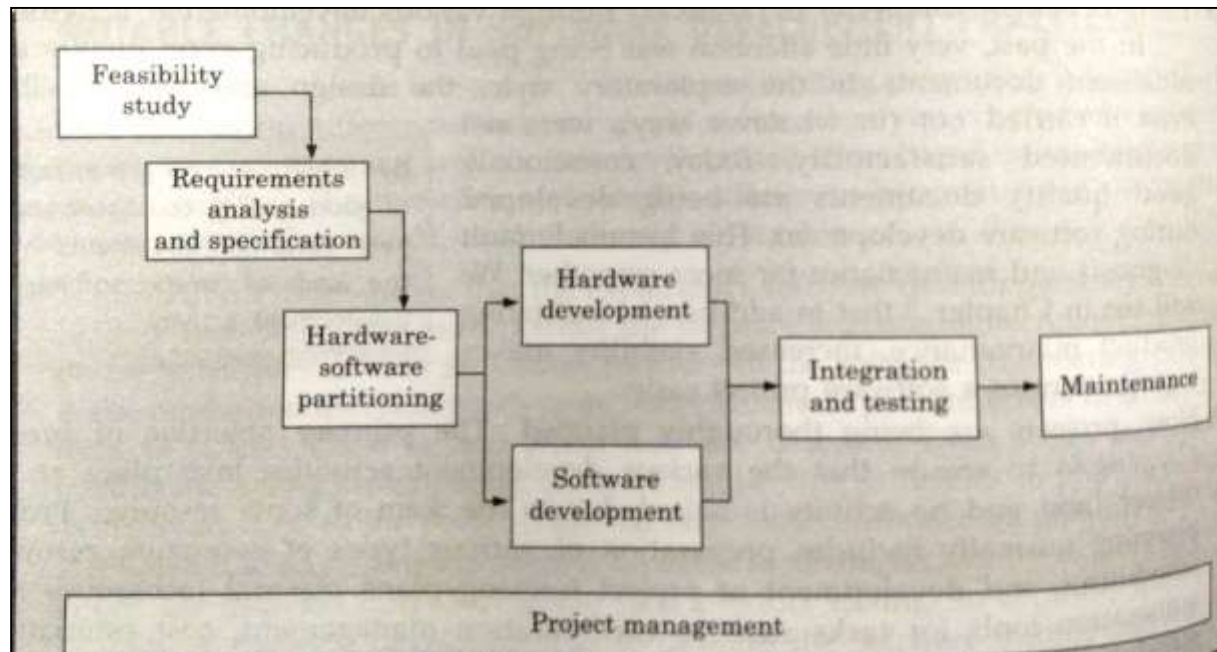# Computer Systems Engineering

- Computer systems engineering:
  - encompasses software engineering.

- Many products require development of software as well as specific hardware to run it:
  - a coffee vending machine,
  - a mobile communication product, etc.

Computer Systems Engineering

- The high-level problem:
  - deciding which tasks are to be solved by software
  - which ones by hardware.

Computer Systems Engineering (CONT.)

- Often, hardware and software are developed together:

  – Hardware simulator is used during software development.

- Integration of hardware and software.

- Final system testing

Evolution of software design techniques over the last 50 years.

- During the 1950s, most programs were being written in assembly language.

- These programs were limited to about a few hundreds of lines of assembly code, i.e. were very small in size.

- **Every programmer developed programs in his own individual style - based on his intuition.**

- This type of programming was called **Exploratory Programming.**

- The next significant development which occurred during early 1960s in the area computer programming was the high-level language programming.

- Use of high-level language programming reduced development efforts and development time significantly.

- Languages like FORTRAN, ALGOL, and COBOL were introduced at that time.

- As the size and complexity of programs kept on increasing, the exploratory programming style proved to be insufficient.

- Programmers found it increasingly difficult not only to write cost-effective and correct programs, but also to understand and maintain programs written by others.

- In late 1960s to cope with this problem, particular attention to the design of the **program's control flow structure was advised**.

- It introduced **"GOTO"** statement which makes control structure of a program complicated and messy.

- The use of "**GOTO**" statements in high-level languages were very natural because of their familiarity with **JUMP** statements which are very frequently used in assembly language programming. But sometimes GOTO is making the loop very complex, and considered harmful.

- Due to the challenges of previous structure it was conclusively proved that only three programming constructs – **sequence, selection, and iteration** – were sufficient to express any programming logic.

- This formed the basis of the **structured programming methodology.**

Control Flow-Based Design (Late 60s)

Structured program

- Three programming
  constructs are sufficient to
  express any programming
  logic:

  - sequence (e.g.
    a=0;b=5;)

  - selection (e.g.if(c=true)
    k=5 else m=5;)

  - iteration (e.g.
    while(k>0) k=j-k;)
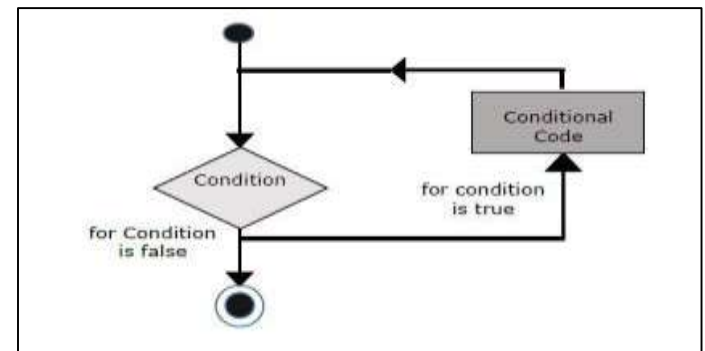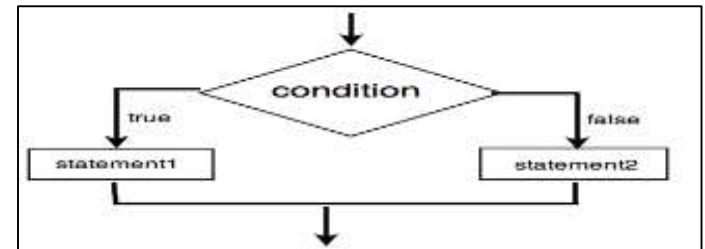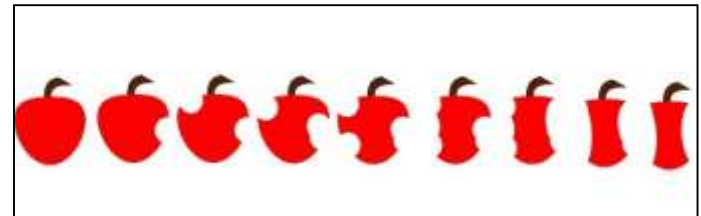
Features of a structured program.

- A structured program uses three types of program constructs i.e. selection, sequence and iteration. Easy to read and understand

- Structured programs avoid unstructured control flows by restricting the use of GOTO statements.

- A structured program consists of a well partitioned set of modules.

- Structured programming uses single entry, single-exit program constructs such as if-then-else, do-while, etc.

- Thus, **the structured programming principle emphasizes designing neat control structures for programs.**

- After structured programming, the next important development was **data structure-oriented design.**

- For writing a good program, it is important to pay more attention to the **design of data structure,** of the program rather than to the design of its control structure.

- **Data structure oriented design** techniques actually help to derive program structure from the data structure of the program.

- Example of a very popular data structure-oriented design technique is **Jackson's Structured Programming (JSP)** methodology, developed by Michael Jackson in the 1970s.

a data structure is a data organization and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data

- Next significant development in the late 1970s was the development of **data flow-oriented design** technique.

- Experienced programmers stated that to have a good program structure, **one has to study how the data flows from input to the output of the program.**

- Every program reads data and then processes that data to produce some output.

- **Once the data flow structure is identified, then from there one can derive the program structure.**

- **Object-oriented design** (1980s) is the latest and very widely used technique.

- It has an intuitively appealing design approach in which natural objects (such as employees, pay-roll register, etc.) occurring in a problem are first identified.

- Relationships among objects (such as composition, reference and inheritance) are determined.

- Each object essentially acts as a data hiding entity.

Evolution of Other Software Engineering Techniques

- life cycle models,
- specification techniques,
- project management techniques,
- testing techniques,
- debugging techniques,
- quality assurance techniques,
- software measurement techniques,
- CASE tools, etc.

Exploratory Programming

Develop outline specification → Build software system → Use software system → System adequate? — NO → Build software system; YES → Deliver software system

Modern Software development process
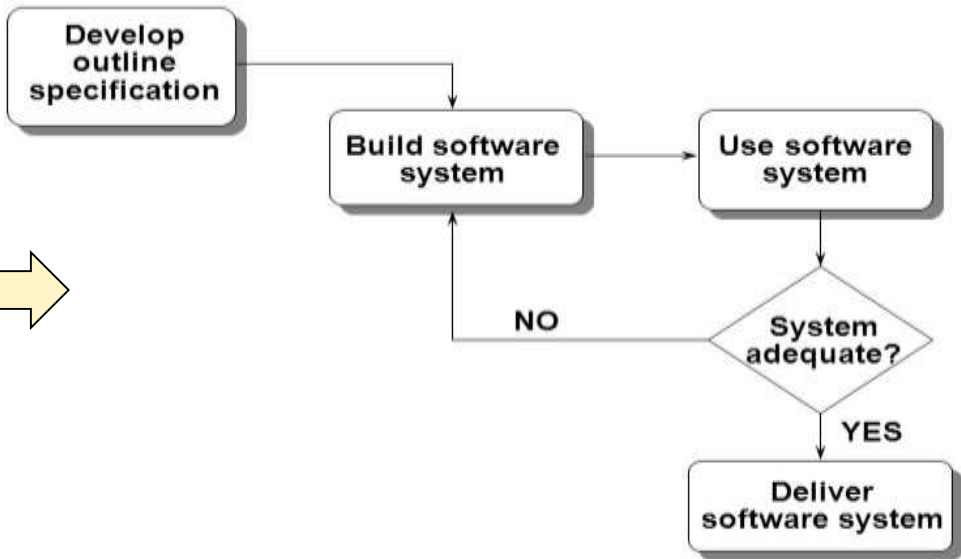
Requirements Analysis → Design → Coding → Testing → Maintenance

Differences between the exploratory style and modern software development practices

- Use of Life Cycle Models
- Software is developed through several well-defined stages:
  - requirements analysis and specification,
  - design,
  - coding,
  - testing, etc.

Differences between the exploratory style and modern software development practices

- Emphasis has shifted
  - from error correction to error prevention.

- Modern practices emphasize:
  - detection of errors as close to their point of introduction as possible.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - errors are detected only during testing,
- Now,
  - focus is on detecting as many errors as possible in each phase of development.

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
  - coding is synonymous with program development.
- Now,
  - coding is considered only a small part of program development effort.

Differences between the exploratory style and modern software development practices (CONT.)

- A lot of effort and attention is now being paid to:
  - requirements specification.
- Also, now there is a distinct design phase:
  - standard design techniques are being used.

Differences between the exploratory style and modern software development practices (CONT.)

- During all stages of development process:
  - Periodic reviews are being carried out

- Software testing has become systematic:
  - standard testing techniques are available.

Differences between the exploratory style and modern software development practices (CONT.)

- There is better visibility of design and code:
  - visibility means production of good quality, consistent and standard documents.
  - In the past, very little attention was being given to producing good quality and consistent documents.
  - The models developed increases visibility which makes software project management easier.

Differences between the exploratory style and modern software development practices (CONT.)

- Because of good documentation:
  - fault diagnosis and maintenance are smoother now.

- Several metrics are being used:
  - help in software project management, quality assurance, etc.

Differences between the exploratory style and modern software development practices (CONT.)

- Projects are being thoroughly planned:
  - estimation,
  - scheduling,
  - monitoring mechanisms.
- Use of CASE tools.

Exploratory style vs. modern style of software development.

Exploratory style	Modern style
Emphasis on error correction	Emphasis on error prevention
Coding was considered synonymous with software development. Believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily.	Coding is regarded as only a small part of the overall software development activities. There are several development activities such as design and testing which typically require much more effort than coding.

Software Life Cycle

- Software life cycle (or software process):
  - series of identifiable stages that a software product undergoes during its life time:
    * Feasibility study
    * requirements analysis and specification,
    * design,
    * coding,
    * testing
    * maintenance.

Life Cycle Model

- A software life cycle model (or process model):
  - a descriptive and diagrammatic model of software life cycle:
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.

Life Cycle Model (CONT.)

- Several different activities may be carried out in each life cycle phase.

  – For example, the design stage might consist of:

    * structured analysis activity followed by
    * structured design activity.

Why Model Life Cycle ?

- A written description:
  - forms a common understanding of activities among the software developers.
  - helps in identifying inconsistencies, redundancies, and omissions in the development process.
  - Helps in tailoring a process model for specific projects.

Why Model Life Cycle ?

- Processes are tailored for special projects.

  - A documented process model

    * helps to identify where the tailoring is to occur.

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model:
  - and then adhere to it.
  - Primary advantage of adhering to a life cycle model:
    * helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
  - he has the freedom to decide his exact steps.

Life Cycle Model (CONT.)

- When a software product is being developed by a team:
  - there must be a precise understanding among team members as to when to do what,
  - otherwise it would lead to chaos and project failure.

Life Cycle Model (CONT.)

- A software project will never succeed if:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure
  - another defines the I/O for his portion first.

Life Cycle Model (CONT.)

- A life cycle model:
  - defines entry and exit criteria for every phase.
  - A phase is considered to be complete:
    - only when all its exit criteria are satisfied.

Life Cycle Model (CONT.)

- The phase exit criteria for the software requirements specification phase:
  - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.

- A phase can start:
  - only if its phase-entry criteria have been satisfied.

Life Cycle Model (CONT.)

- It becomes easier for software project managers:
  - to monitor the progress of the project.

Life Cycle Model (CONT.)

- When a life cycle model is adhered to,
  - the project manager can at any time fairly accurately tell,
    * at which stage (e.g., design, code, test, etc.) of the project is.
  - Otherwise, it becomes very difficult to track the progress of the project
    * the project manager would have to depend on the guesses of the team members.

57

Life Cycle Model (CONT.)

- This usually leads to a problem:
  - known as the 99% complete syndrome.

Life Cycle Model (CONT.)

- Many life cycle models have been proposed.
- We will confine our attention to a few important and commonly used models.
  - classical waterfall model
  - iterative waterfall,
  - evolutionary,
  - prototyping, and
  - spiral model

Summary

- Software engineering is:
  - systematic collection of decades of programming experience
  - together with the innovations made by researchers.

Summary

- A fundamental necessity while developing any large software product:
  - adoption of a life cycle model.

Summary

- Adherence to a software life cycle model:
  - helps to do various development activities in a systematic and disciplined manner.
  - also makes it easier to manage a software development effort.

Reference

- R. Mall, "Fundamentals of Software Engineering," Prentice-Hall of India, 1999, <u>CHAPTER 1.</u>