Introduction

Dr. RAJIB MALL

Professor

Department Of Computer Science & Engineering

IIT Kharagpur.

About Myself

- RAJIB MALL
- B.E. , M.E., Ph.D from Indian Institute of Science, Bangalore
- Worked with Motorola (India)
- Shifted to IIT, Kharagpur in 1994
  - Currently Professor at CSE department

What is Software Engineering?

- Engineering approach to develop software.
  - –Building Construction Analogy.
- Systematic collection of past experience:
  - –Techniques,
  - –Methodologies,
  - –Guidelines.

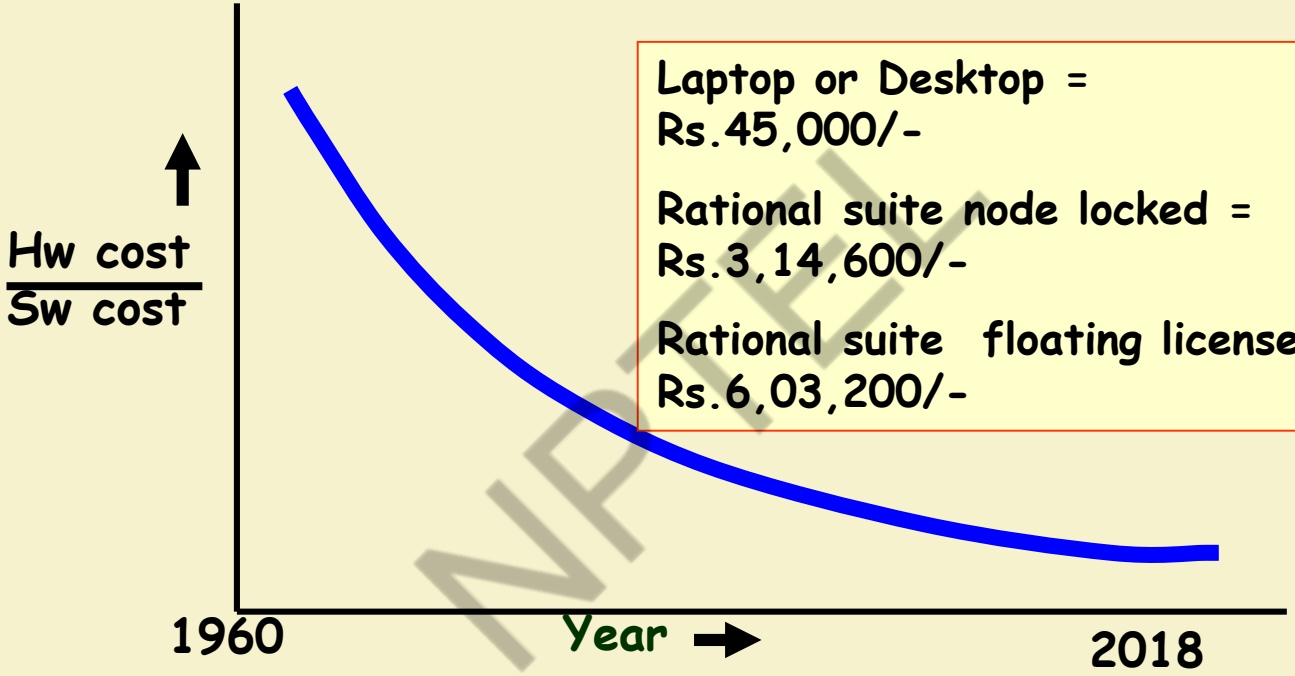
IEEE Definition

- "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

Software Crisis

- It is often the case that software products:
  - Fail to meet user requirements.
  - Expensive.
  - Difficult to alter, debug, and enhance.
  - Often delivered late.
  - Use resources non-optimally.

Software Crisis (cont.)



Laptop or Desktop = Rs.45,000/-

Rational suite node locked = Rs.3,14,600/-

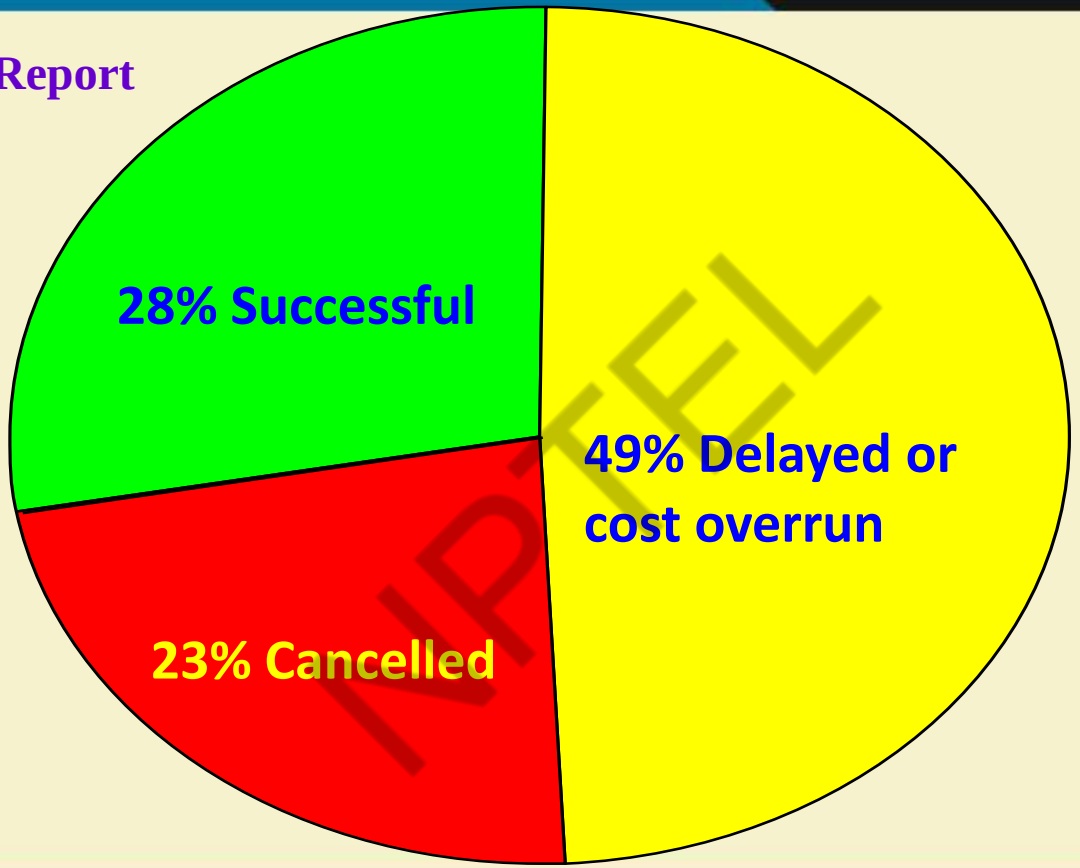
Rational suite floating license= Rs.6,03,200/-

Relative Cost of Hardware and Software

Then why not have entirely hardware systems?...

- A virtue of software:
  - Relatively easy and faster to develop and to change…
  - Consumes no space, weight, or power…
  - Otherwise all might as well be hardware.
- The more is the complexity of software, the harder it is to change--why?
  - Further, the more the changes made to a program, the greater becomes its complexity.

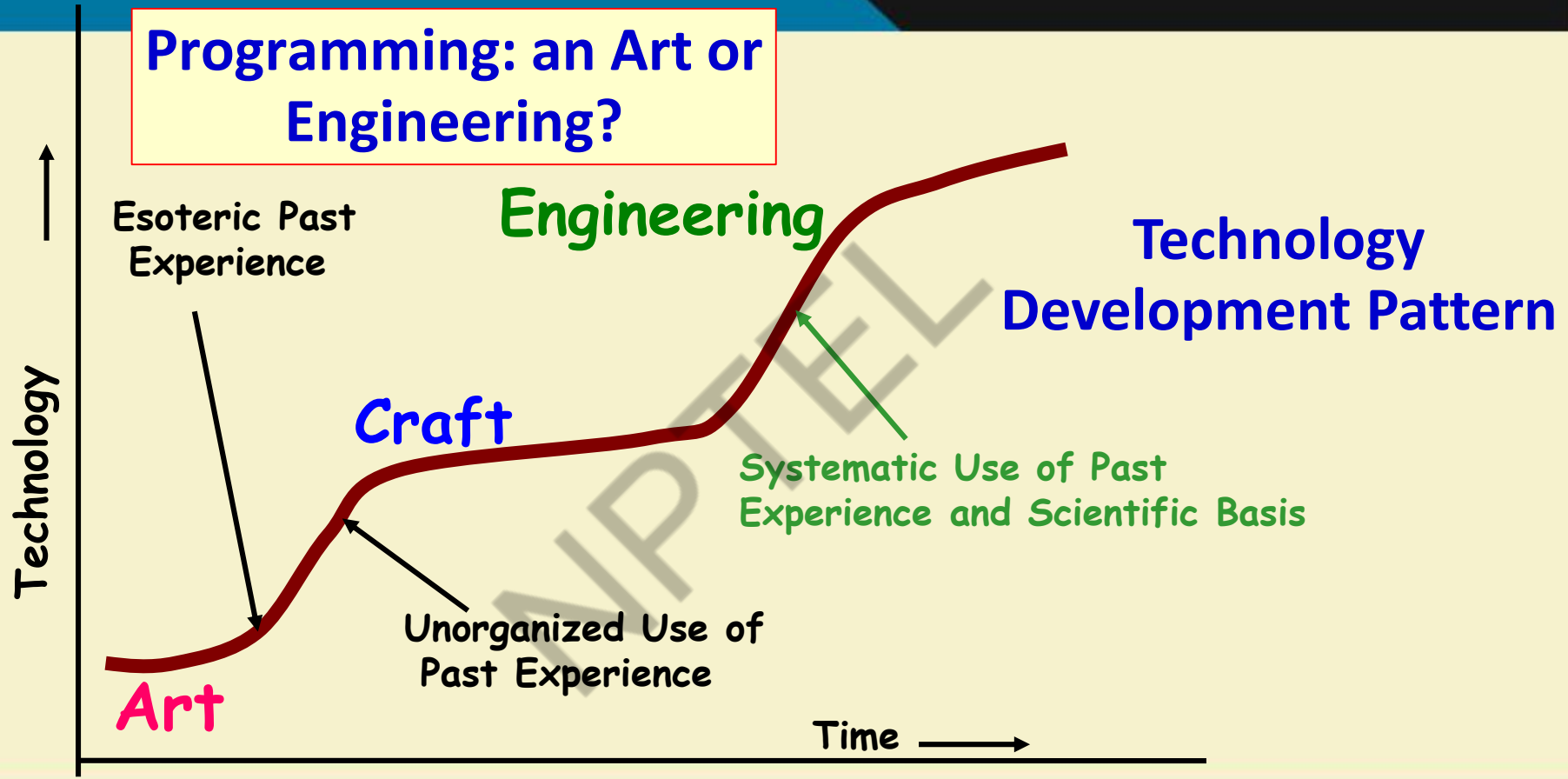
Standish Group Report



Which Factors are Contributing to the Software Crisis?

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

Programming: an Art or Engineering?



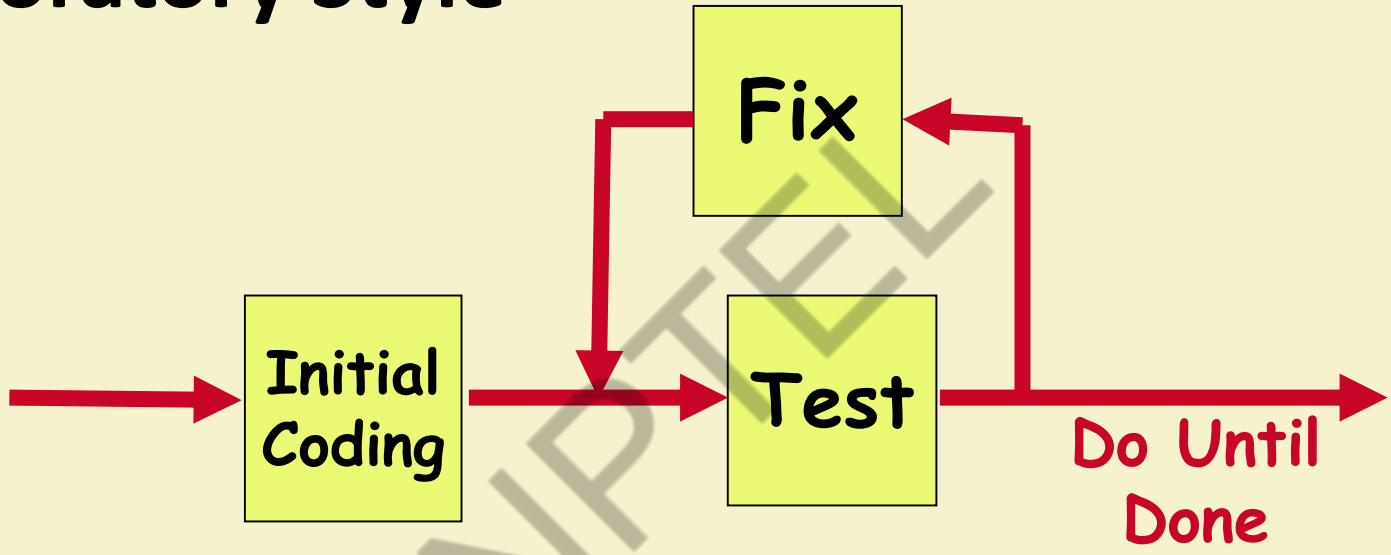
Programming an Art or Engineering?

- Heavy use of past experience:
  - Past experience is systematically arranged.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives.
- Pragmatic approach to cost-effectiveness.

- Early programmers used **exploratory** (also called build and fix) style.
  - A `dirty' program is quickly developed.
  - The bugs are fixed as and when they are noticed.
  - Similar to how a junior student develops programs…

What is Exploratory Software Development?

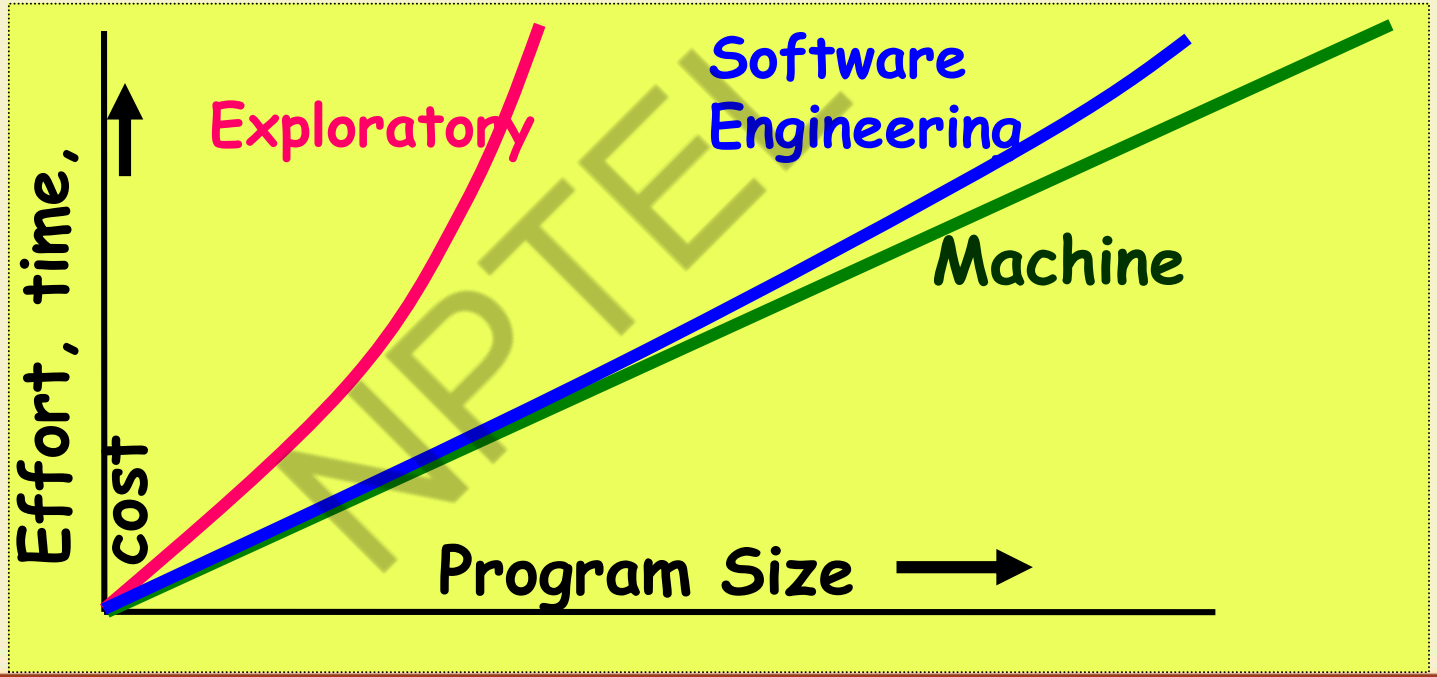
Exploratory Style



Does not work for nontrivial projects… Why?...

What is Wrong with the Exploratory Style?

- Can successfully be used for developing only very small (toy) programs.



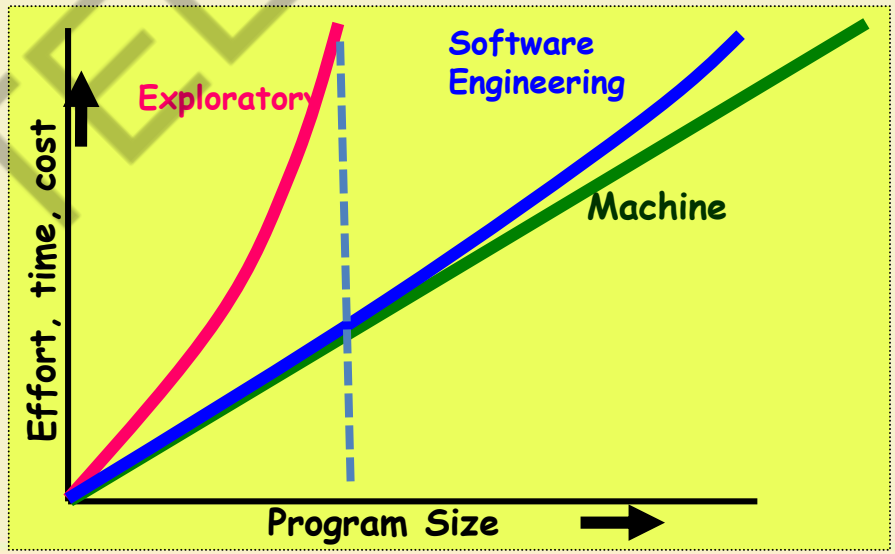
What is Wrong with the Exploratory Style?

Cont…

- Besides the exponential growth of effort, cost, and time with problem size:
  - Exploratory style usually results in unmaintainable code.
  - **It becomes very difficult to use the exploratory style in team development environments…**

What is Wrong with the Exploratory Style? Cont...

- **Why does the effort required to develop a software grow exponentially with size?**
- Why does the approach completely breaks down when the size of software becomes large?



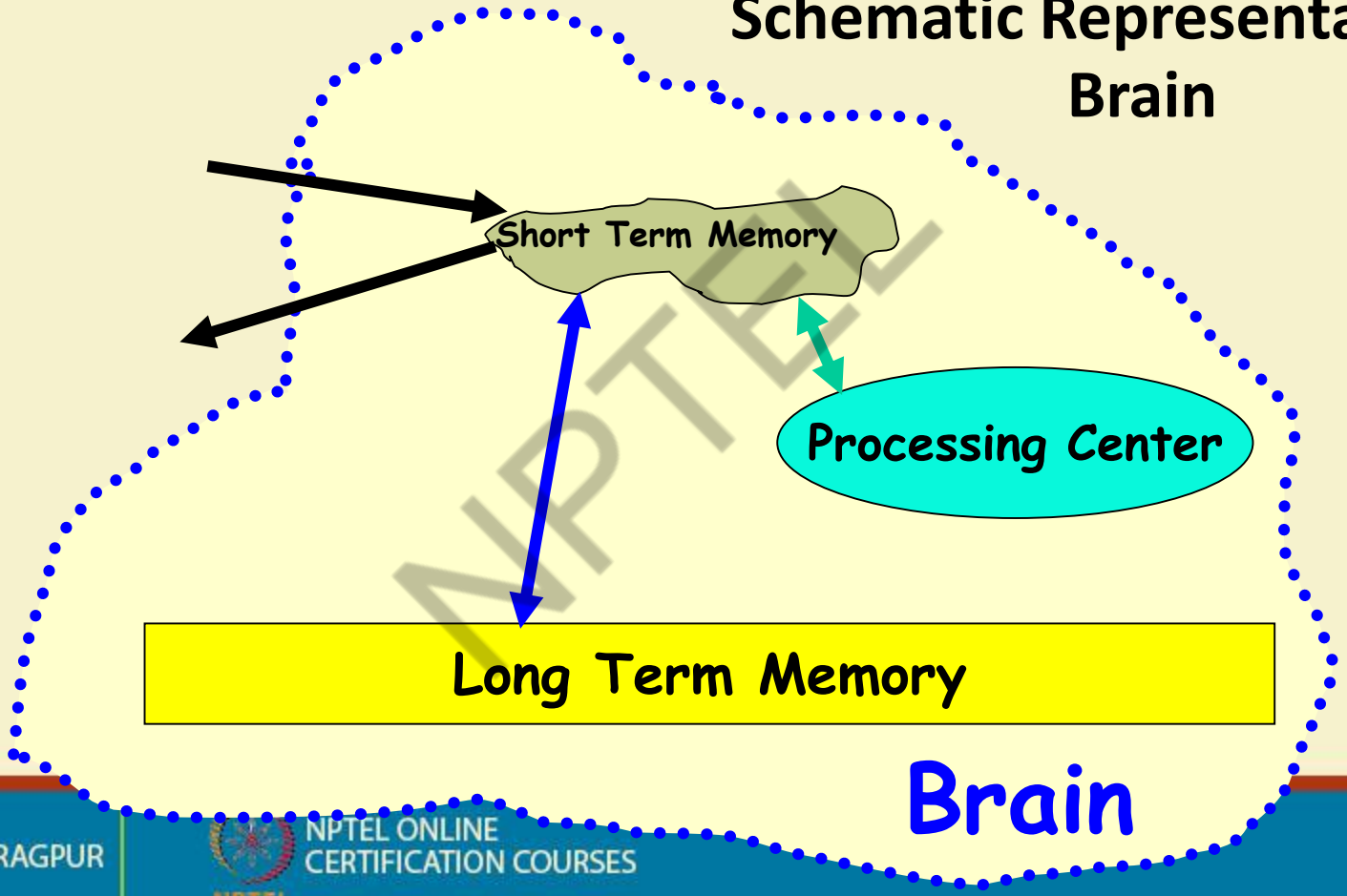
An Interpretation Based on Human Cognition Mechanism

- Human memory can be thought to be made up of two distinct parts [Miller 56]:
  - **Short term memory and**
  - **Long term memory.**

Human Cognition Mechanism

- Suppose I ask: **“It is 10:10AM now, how many hours are remaining today?"**
  - 10AM would be stored in the short-term memory.
  - “A day is 24 hours long.” would be fetched from the long term memory into short term memory.
  - The mental manipulation unit would compute the difference (24-10).

Schematic Representation of Brain



- An item stored in the short term memory can get lost:
  - **Either due to decay with time or**
  - **Displacement by newer information.**

Short Term Memory

- This restricts the time for which an item is stored in short term memory:
  - Typically few tens of seconds.
  - However, an item can be retained longer in the short term memory by recycling.

What is an Item?

- **An item is any set of related information.**
  - A character such as `a' or a digit such as `5'.
  - A word, a sentence, a story, or even a picture.
- Each item normally occupies one place in memory.
- When you are able to relate several different items together (chunking):
  - The information that should normally occupy several places, takes only one place in memory.

Chunking

- If I ask you to remember the number **110010101001**
  - It may prove very hard for you to understand and remember.
  - But, the octal form of **6251** **(110)(010)(101)(001)** would be easier.
  - You have managed to create chunks of three items each.

Evidence of Short Term Memory

- In many of our day-to-day experiences:
  - **Short term memory is evident.**
- Suppose, you look up a number from the telephone directory and start dialling it.
  - If you find the number is busy, you can dial the number again after a few seconds without having to look up the number from directory.
- But, after several days:
  - You may not remember the number at all
  - Would need to consult the directory again.

- If a person deals with seven or less number of items:
  - These would be accommodated in the short term memory.
  - So, he can easily understand it.

The Magical Number 7

- As the number of new information increases beyond seven:
  - It becomes exceedingly difficult to understand it.

What is the Implication in Program Development?

- A small program having just a few variables:
  - Is within easy grasp of an individual.
- As the number of independent variables in the program increases:
  - **It quickly exceeds the grasping power of an individual…**
  - **Requires an unduly large effort to master the problem.**

Implication in Program Development

- Instead of a human, if a machine could be writing (generating) a program,
  - The slope of the curve would be linear.
- But, how does use of software engineering principles helps hold down the effort-size curve to be almost linear?
  - **Software engineering principles extensively use techniques specifically targeted to overcome the human cognitive limitations.**

Which Principles are Deployed by Software Engineering Techniques to Overcome Human Cognitive Limitations?

- Two important principles are profusely used:
  - **Abstraction**
  - **Decomposition**

Two Fundamental Techniques to Handle Complexity

What is Abstraction?

- Simplify a problem by omitting unnecessary details.
  - **Focus attention on only one aspect of the problem and ignore other aspects and irrelevant details.**
  - Also called model building.

- Suppose you are asked to develop an overall understanding of some country.

Abstraction Example

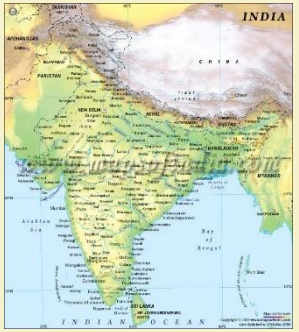
  - Would you:
    - Meet all the citizens of the country, visit every house, and examine every tree of the country?
  - You would possibly refer to various types of maps for that country only.

You would study an Abstraction…

- A map is:
  - An abstract representation of a country.
  - Various types of maps (abstractions) possible.

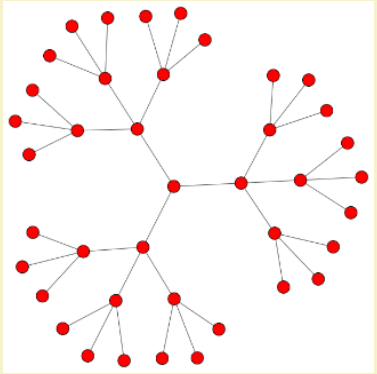
Does every Problem have a single Abstraction?

- Several abstractions of the same problem can be created:
  - Focus on some specific aspect and ignore the rest.
  - Different types of models help understand different aspects of the problem.



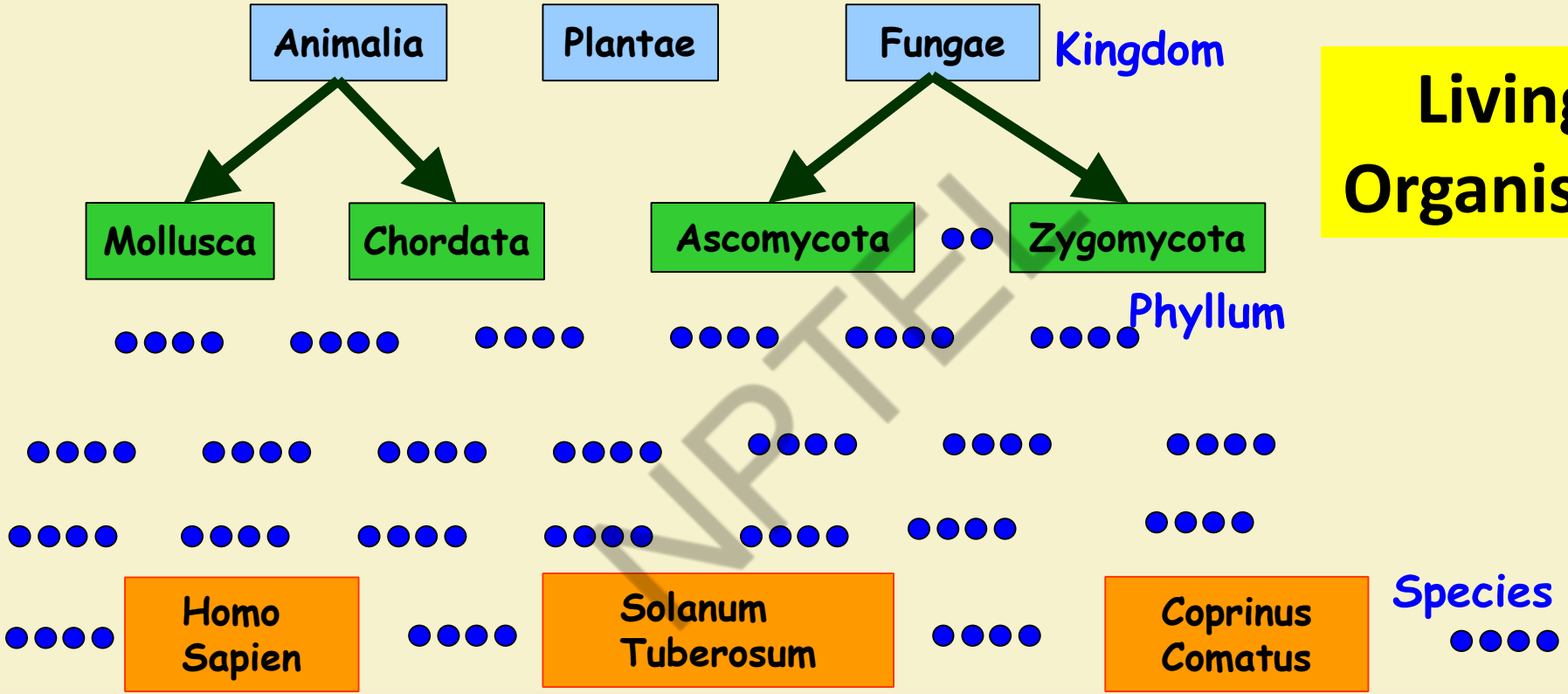
Abstractions of Complex Problems

- For complex problems:
  - A single level of abstraction is inadequate.
  - A hierarchy of abstractions may have to be constructed.
- Hierarchy of models:
  - A model in one layer is an abstraction of the lower layer model.
  - An implementation of the model at the higher layer.



Abstraction of Complex Problems -- An Example

- Suppose you are asked to understand all life forms that inhabit the earth.
- Would you start examining each living organism?
  - You will almost never complete it.
  - Also, get thoroughly confused.
- **Solution: Try to build an abstraction hierarchy.**



Living Organisms

Quiz

- What is a model?
- Why develop a model? That is, how does constructing a model help?
- Give some examples of models.

Decomposition

- Decompose a problem into many small independent parts.
  - The small parts are then taken up one by one and solved separately.
  - **The idea is that each small part would be easy to grasp and therefore can be easily solved.**
  - **The full problem is solved when all the parts are solved.**

Decomposition

- A popular example of decomposition principle:
  - Try to break a bunch of sticks tied together versus breaking them individually.

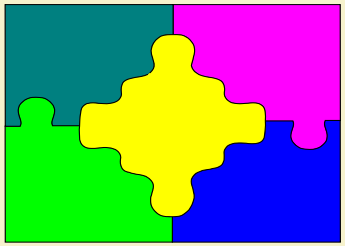
- Any arbitrary decomposition of a problem may not help.
  - The decomposed parts must be more or less independent of each other.

Decomposition: Another Example

- Example use of decomposition principle:
  - You understand a book better when the contents are organized into independent chapters.
  - Compared to when everything is mixed up.

Why Study Software Engineering? (1)

- To acquire skills to develop large programs.
  - **Handling exponential growth in complexity with size.**
  - Systematic techniques based on abstraction (modelling) and decomposition.

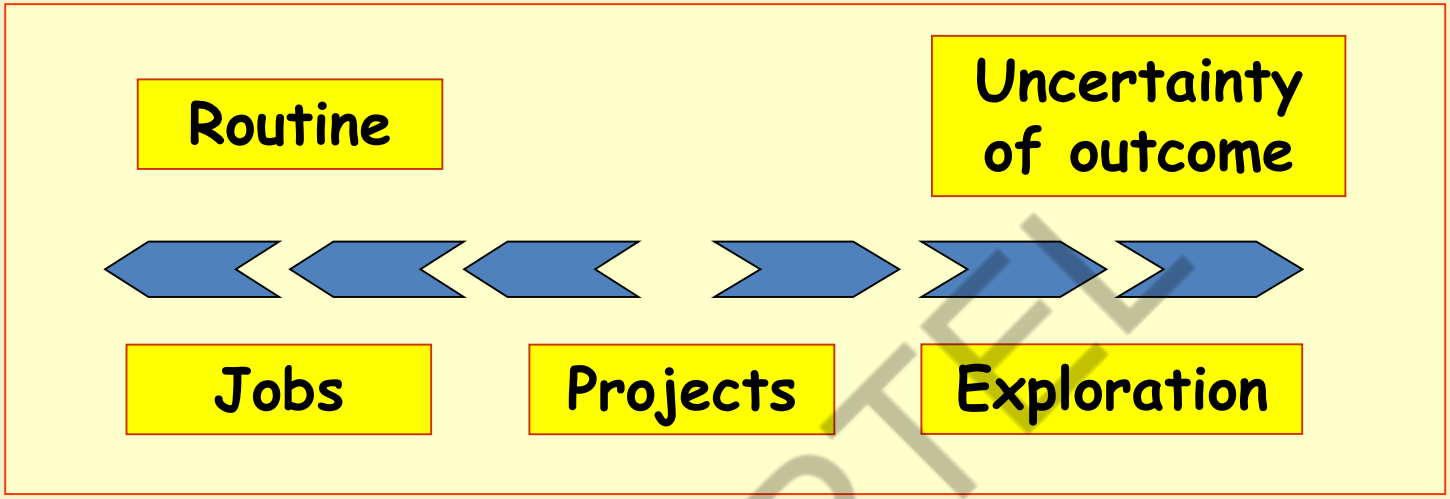


Why Study Software Engineering? (2)

- Learn systematic techniques of:
  - **Specification, design, user interface development, testing, project management, maintenance, etc.**
  - Appreciate issues that arise in team development.

Why Study Software Engineering? (3)

- To acquire skills to be a better programmer:
  - Higher Productivity
  - Better Quality Programs



Jobs versus Projects

Jobs – repetition of very well-defined and well understood tasks with very little uncertainty

Exploration – The outcome is very uncertain, e.g. finding a cure for cancer.

Projects – in the middle! Has challenge as well as routine…

- Two types of software projects:
  - **Products (Generic software)**
  - **Services (custom software)**

Types of Software Projects

- Total business – Several Trillions of US $
  - Half in products and half services
  - **Services segment is growing fast!**

Types of Software

Packaged software – prewritten software available for purchase

- **Horizontal market software**—meets needs of many companies
- **Vertical market software**—designed for particular industry

Custom software – software developed at some user's requests-Usually developer tailors some generic solution

Types of Software Projects

- Software product development projects
- Software services projects

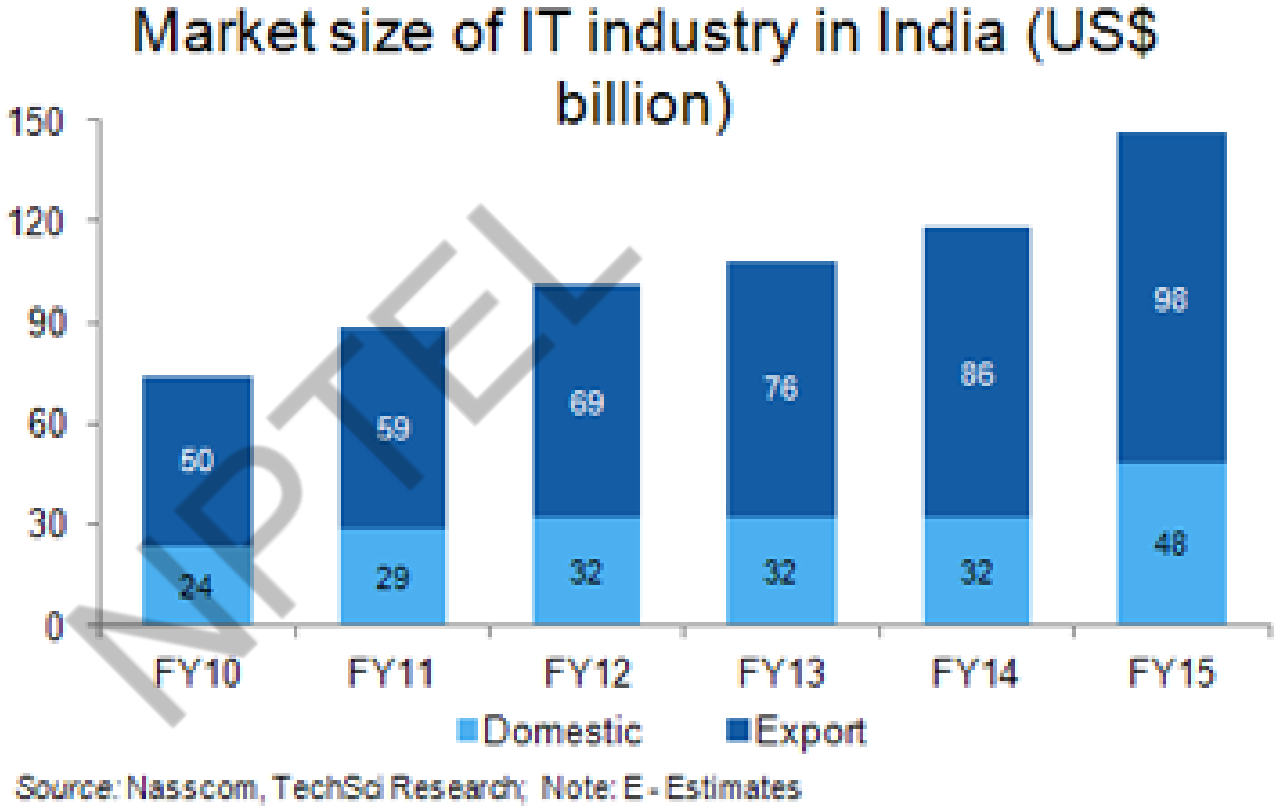
Software Services

- Software service is an umbrella term, includes:
  - Software customization
  - Software maintenance
  - Software testing
  - Also contract programmers (CP) carrying out coding or any other assigned activities.

Factors responsible for accelerated growth of services…

- Now lots of code is available in a company:
  - New software can be developed by modifying the closest.
- Speed of Conducting Business has increased tremendously:
  - Requires shortening of project duration

Contribution of the IT sector to India's GDP rose to approximately 9.5% in 2015 from 1.2% in 98



Scenario of Indian Software Companies

- Indian companies have largely focused on the services segment -- - Why?

A Few Changes in Software Project Characteristics over Last 40 Years

- 40 years back, very few software existed
  - **Every project started from scratch**
  - **Projects were multi year long**
- The programming languages that were used earlier hardly provided any scope for reuse:
  - FORTRAN, PASCAL, COBOL, BASIC
- No application was GUI-based:
  - Mostly command selection from displayed text menu items.

Traditional versus Modern Projects

- Projects are increasingly becoming services:
  - Either tailor some existing software or reuse pre-built libraries.
- Facilitate and accommodate client feedbacks
- Facilitate customer participation in project development work
- Incremental software delivery with evolving functionalities.
- **No software is being developed from scratch --- Significant reuse is being made…**

Computer Systems Engineering

- **Many products require development of software as well as specific hardware to run it:**
  - a coffee vending machine,
  - a robotic toy,
  - A new health band product, etc.
- Computer systems engineering:
  - encompasses software engineering.

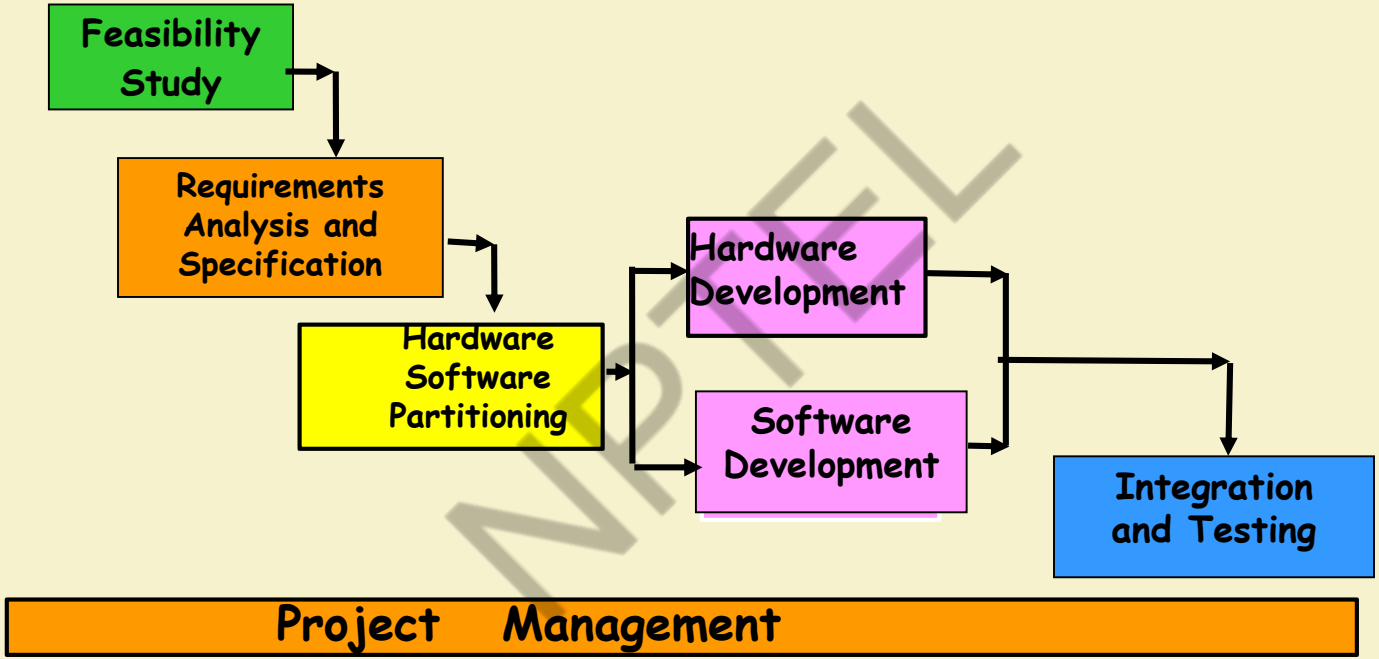
Computer Systems Engineering

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

Computer Systems Engineering (CONT.)

- Typically, hardware and software are developed together:
  - Hardware simulator is used during software development.
- Integration of hardware and software.
- Final system testing

Computer Systems Engineering (CONT.)



Emergence of Software Engineering Techniques

Emergence of Software Engineering Techniques

- Early Computer Programming (1950s):
  - Programs were being written in assembly language…
  - Sizes limited to about a few hundreds of lines of assembly code…

Early Computer Programming (50s)

- Every programmer developed his/her own style of writing programs:
  - –According to his intuition (called exploratory or build-and-fix programming) .

High-Level Language Programming (Early 60s)

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
  - This reduced software development efforts greatly.
  - Why reduces?

High-Level Language Programming (Early 60s)

- **Software development style was still exploratory.**
  - Typical program sizes were limited to a few thousands of lines of source code.

Control Flow-Based Design (late 60s)

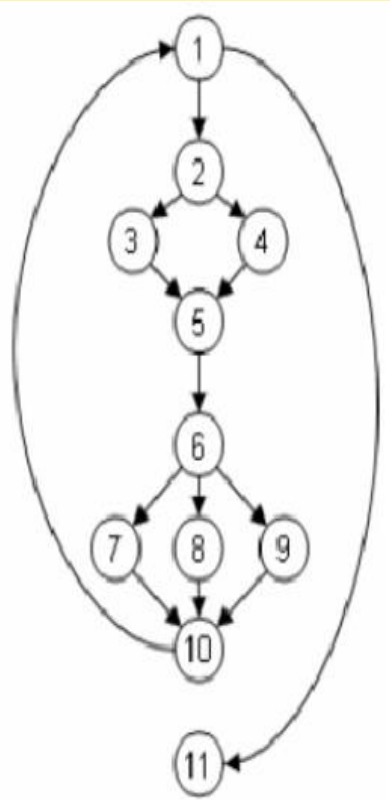
- Size and complexity of programs increased further:
  - –Exploratory programming style proved to be insufficient.
- Programmers found:
  - –Very difficult to write cost-effective and correct programs.

Control Flow-Based Design (late 60s)

- Programmers found it very difficult:
  - –To understand and maintain programs written by others.
- To cope up with this problem, experienced programmers advised--**--"Pay particular attention to the design of the program's control structure."**

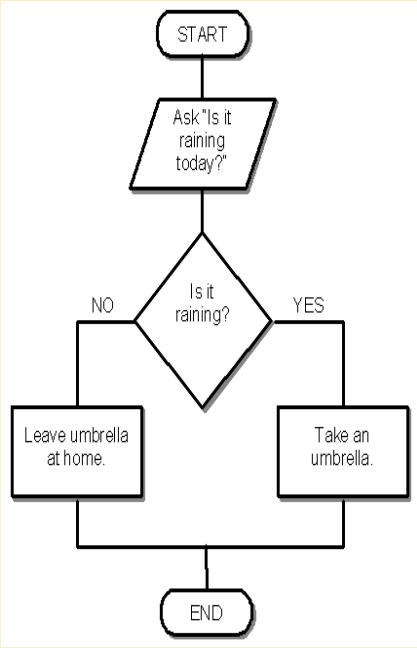
Control Flow-Based Design (late 60s)

- What is a program's control structure?
  - **The sequence in which the program's instructions are executed.**
- To help design programs having good control structure:
  - Flow charting technique was developed.



Control Flow-Based Design (late 60s)

- Using flow charting technique:
  - –One can represent and design a program's control structure.
  - –When asked to understand a program:
    - One would mentally trace the program's execution sequence.

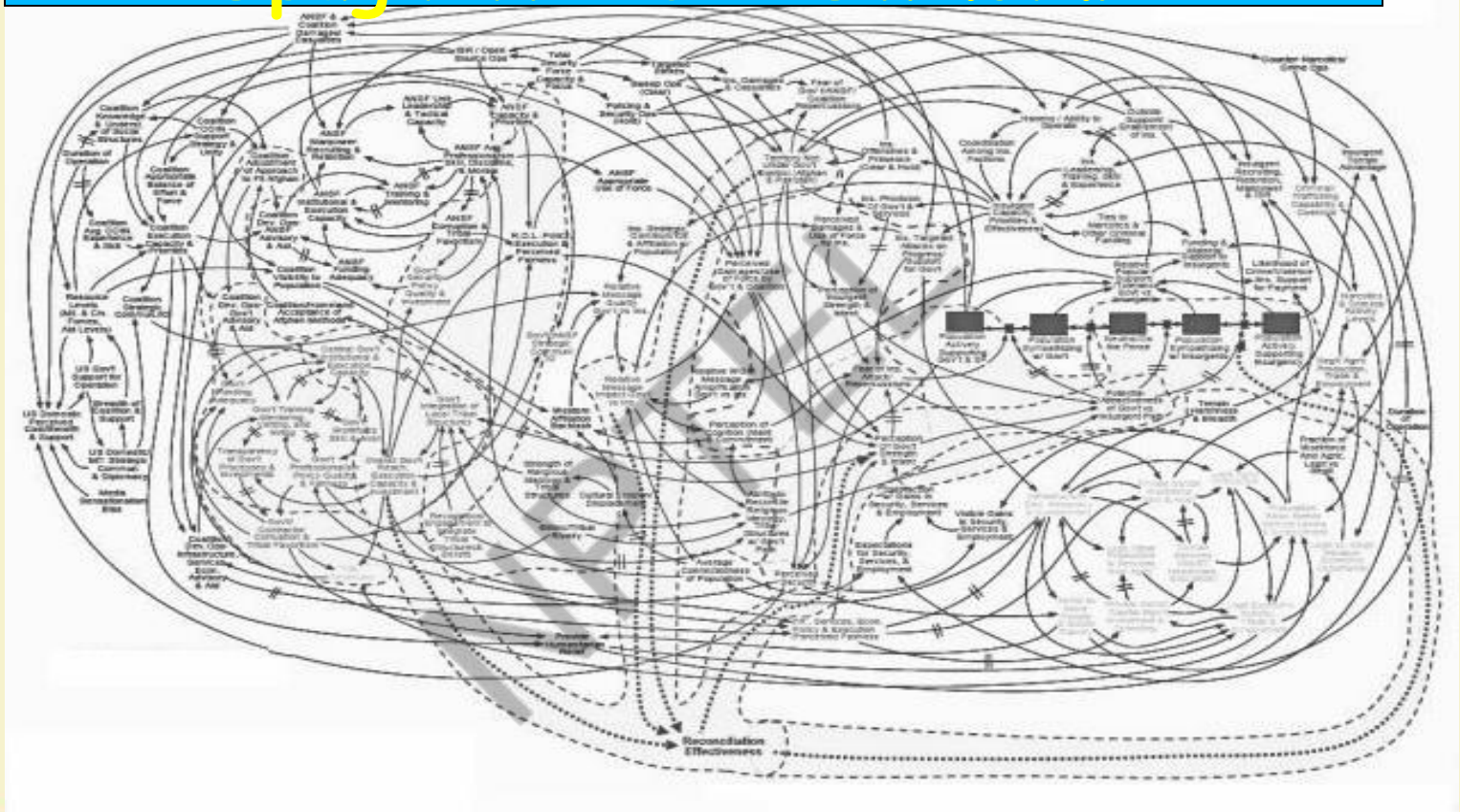


Control Flow-Based Design

- A program having a messy flow chart representation:

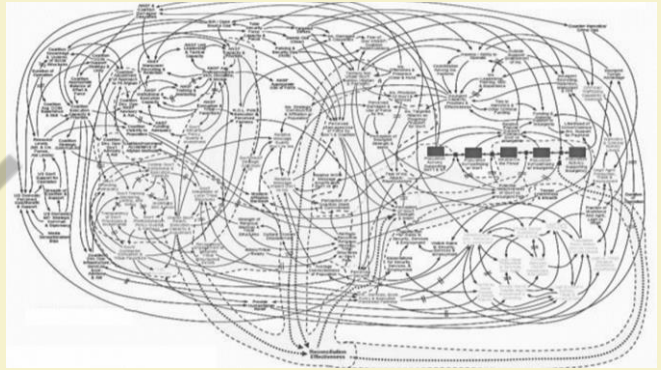
  –**Difficult to understand and debug.**

Spaghetti Code Structure



Control Flow-Based Design (Late 60s)

- What causes program complexity?



  - GO TO statements makes control structure of a program messy.
  - GO TO statements alter the flow of control arbitrarily.
  - The need to restrict use of GO TO statements was recognized.

Control Flow-Based Design (Late 60s)

- Many programmers had extensively used assembly languages.
  - –JUMP instructions are frequently used for program branching in assembly languages.
  - –Programmers considered use of GO TO statements inevitable.

```
addi $a0, $0, 1
j next
next:
j skip1
add $a0, $a0, $a0
skip1:
j skip2
add $a0, $a0, $a0
add $a0, $a0, $a0
skip2:
j skip3
loop:
add $a0, $a0, $a0
add $a0, $a0, $a0
add $a0, $a0, $a0
skip3:
j loop
```

Control-flow Based Design (Late 60s)

- At that time, Dijkstra published his article:
  - "Goto Statement Considered Harmful" Comm. of ACM, 1969.
- Many programmers were unhappy to read his article.

Control Flow-Based Design (Late 60s)

- Some programmers published several counter articles:
  - Highlighted the advantages and inevitability of GO TO statements.

Control Flow-Based Design (Late 60s)

- It soon was conclusively proved:
  - Only three programming constructs are sufficient to express any programming logic:
    - **sequence (a=0;b=5;)**
    - **selection (if(c==true) k=5 else m=5;)**
    - **iteration (while(k>0) k=j-k;)**

Control-flow Based Design (Late 60s)

- Everyone accepted:
  - It is possible to solve any programming problem without using GO TO statements.
  - This formed the basis of **Structured Programming methodology.**

Structured Programming

- A program is called structured:
  - When it uses only the following types of constructs:
    - **sequence,**
    - **selection,**
    - **iteration**
  - Consists of modules.

Structured Programs

- Sometimes, violations to structured programming are permitted:
  - Due to practical considerations such as:
  - Premature loop exit (break) or for exception handling.

Advantages of Structured programming

- Structured programs are:
  - **Easier to read and understand,**
  - **Easier to maintain,**
  - **Require less effort and time for development.**
  - **Less buggy**

Structured Programming

- Research experience shows:

  –Programmers commit less number of errors:

  - While using structured if-then-else and do-while statements.
  - Compared to test-and-branch (GOTO) constructs.

Data Structure-Oriented Design (Early 70s)

- As program sizes increased further, soon it was discovered:
  - **It is important to pay more attention to the design of data structures of a program**
    - Than to the design of its control structure.

Data Structure-Oriented Design (Early 70s)

- Techniques which emphasize designing the data structure:
  - –Derive program structure from it:
    - Are called **data structure-oriented design techniques.**

Data Structure Oriented Design (Early 70s)

- An example of data structure-oriented design technique:
  - Jackson's Structured Programming(JSP) methodology
    - Developed by Michael Jackson in 1970s.

Data Structure Oriented Design (Early 70s)

- **JSP technique:**
  - Program code structure should correspond to the data structure.

A Data Structure Oriented Design
(Early 70s)

- **JSP methodology:**
  - A program's data structures are first designed using notations for
    - **sequence, selection, and iteration.**
  - The data structure design is then used :
    - To derive the program structure.

Data Structure Oriented Design (Early 70s)

- Several other data structure-oriented Methodologies also exist:
  - e.g., Warnier-Orr Methodology.

Data Flow-Oriented Design (Late 70s)

- Data flow-oriented techniques advocate:
  - The data items input to a system must first be identified,
  - Processing required on the data items to produce the required outputs should be determined.

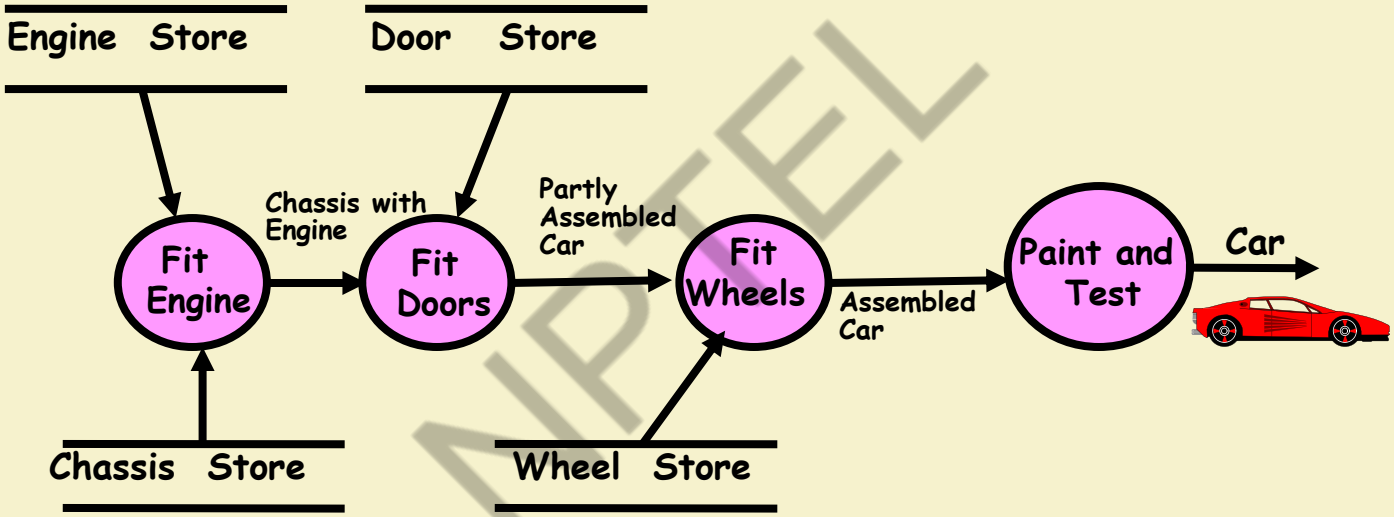
Data Flow-Oriented Design (Late 70s)

- Data flow technique identifies:
  - Different processing stations (functions) in a system.
  - The items (data) that flow between processing stations.

Data Flow-Oriented Design (Late 70s)

- Data flow technique is a generic technique:
  - Can be used to model the working of any system.
    - not just software systems.
- A major advantage of the data flow technique is its simplicity.

Data Flow Model of a Car Assembly Unit



Object-Oriented Design (80s)

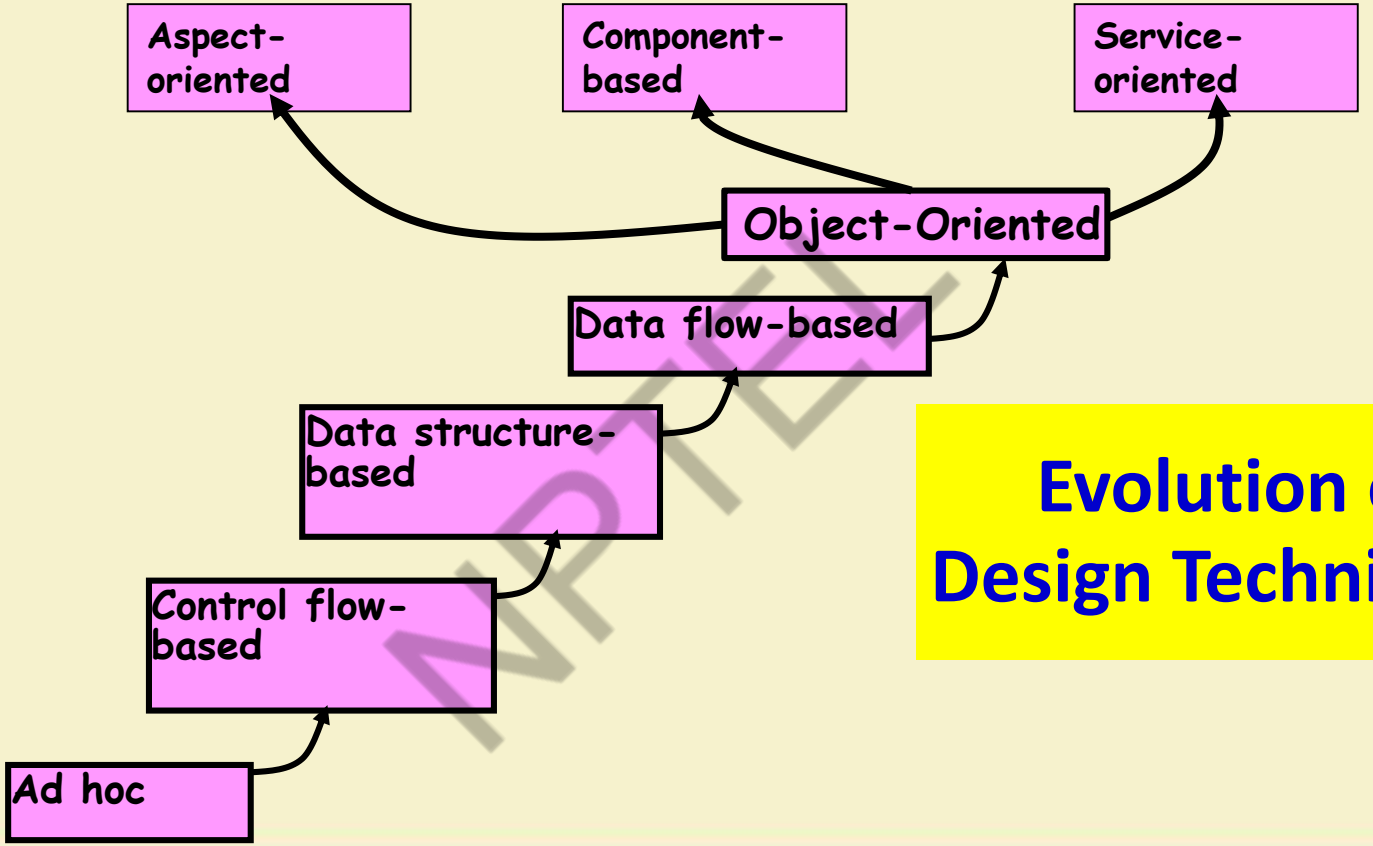
- Object-oriented technique:
  - An intuitively appealing design approach:
  - Natural objects (such as employees, pay-roll-register, etc.) occurring in a problem are first identified.

Object-Oriented Design (80s)

- Relationships among objects:
  - –Such as composition, reference, and inheritance are determined.
- Each object essentially acts as:
  - –A data hiding (or data abstraction) entity.

- Object-Oriented Techniques have gained wide acceptance:
  - Simplicity
  - Increased Reuse possibilities
  - Lower development time and cost
  - More robust code
  - Easy maintenance

Object-Oriented Design (80s)



Evolution of Design Techniques

Evolution of Other Software Engineering Techniques

- The improvements to the software design methodologies
  - –are indeed very conspicuous.
- In additions to the software design techniques:
  - –Several other techniques evolved.

- Life cycle models,
- Specification techniques,
- Project management techniques,
- Testing techniques,
- Debugging techniques,
- Quality assurance techniques,
- Metrics,
- CASE tools, etc.

Evolution of Other Software Engineering Techniques

- 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

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.
  - We will see later that increased visibility 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.)

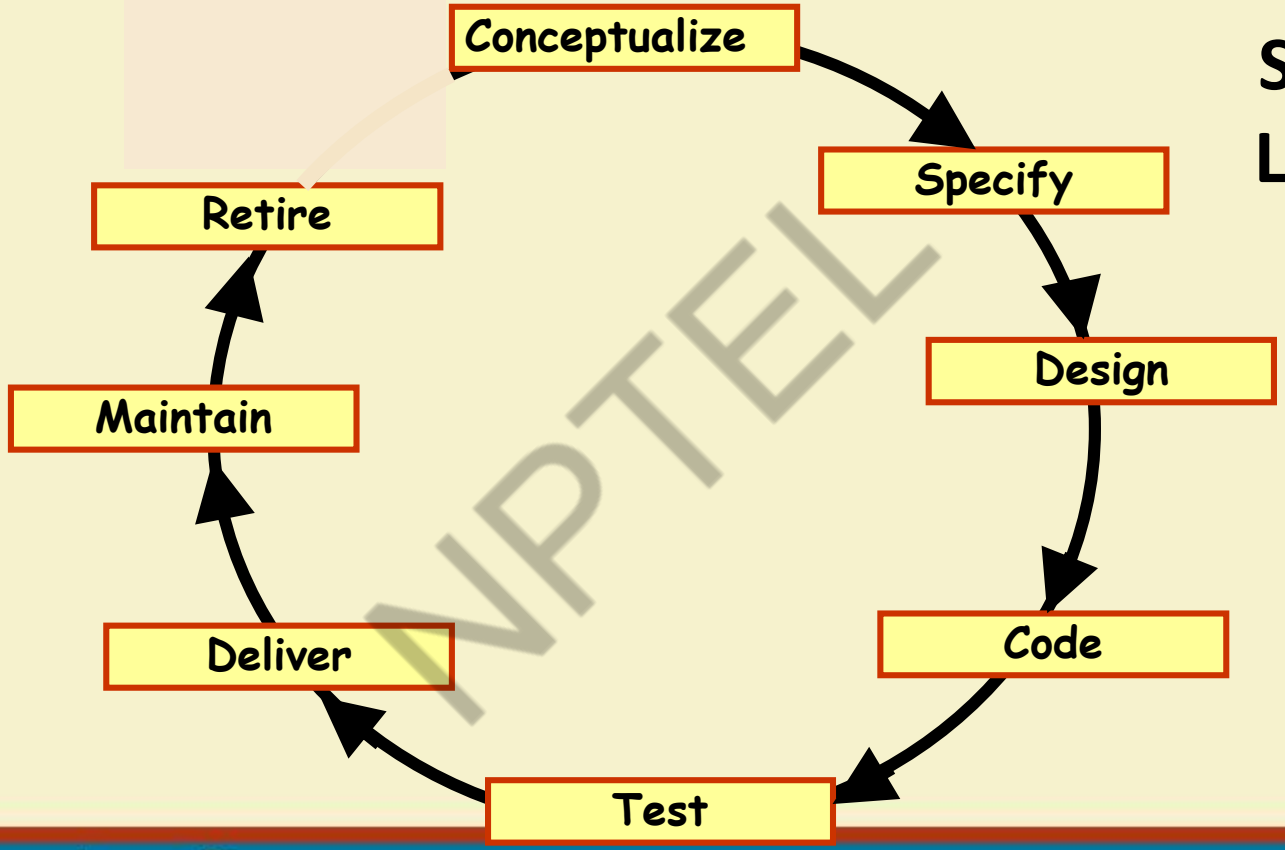
- Projects are being properly planned:
  - –estimation,
  - –scheduling,
  - –monitoring mechanisms.
- Use of CASE tools.

Review Questions

- What is structured programming?
- What problems may appear if a large program is developed without using structured programming techniques?

Life Cycle Models

Software Life Cycle



Life Cycle Model

- A software life cycle model (also process model or SDLC):
  - A descriptive and diagrammatic model of software life cycle:
  - Identifies all the activities undertaken during product development,
  - Establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.

Life Cycle Model (CONT.)

- Each life cycle phase consists of several activities.
  - For example, the design stage might consist of:
    - structured analysis
    - structured design
    - Design review

Why Model Life Cycle?

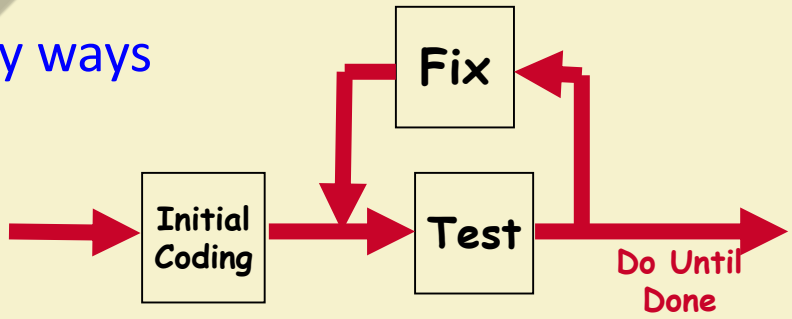
- A graphical and written description:
  - **Helps common understanding of activities among the software developers.**
  - **Helps to identify inconsistencies, redundancies, and omissions in the development process.**
  - **Helps in tailoring a process model for specific projects.**

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 ---
  - The problem is within the grasp of an individual.
  - He has the freedom to decide his exact steps and still succeed --- called Exploratory model--- One can use it in many ways
  - Code → Test → Design
  - Code → Design → Test → Change Code →
  - Specify → code → Design → Test → etc.



Life Cycle Model (CONT.)

- When software 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.

Phase Entry and Exit Criteria

- 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.)

- What is 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: Milestones

- Milestones help software project managers:
  - –Track the progress of the project.
  - –Phase entry and exit are important milestones.

Life Cycle and Project Management

- When a life cycle model is followed:
  - The project manager can at any time fairly accurately tell,
    - At which stage (e.g., design, code, test, etc.) the project is.

Project Management Without Life Cycle Model

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

Project Deliverables: Myth and Reality

Myth:

The only deliverable for a successful project is the working program.

Reality:

Documentation of all aspects of software development are needed to help in operation and maintenance.

Life Cycle Model (CONT.)

- Many life cycle models have been proposed.
- We confine our attention to only a few commonly used models.
  - Waterfall
  - V model,
  - Evolutionary,
  - Prototyping
  - Spiral model,
  - Agile models

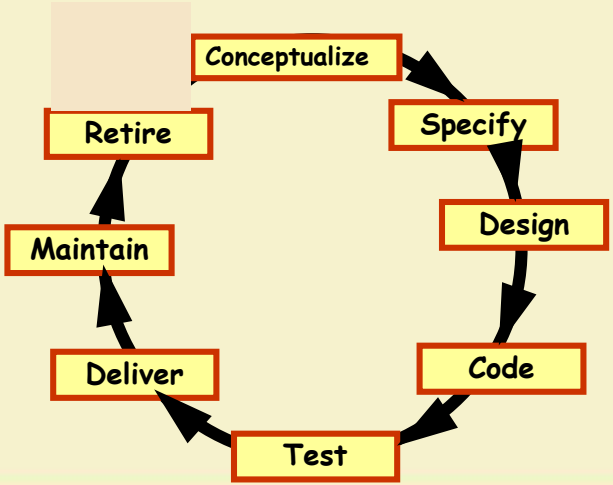
Traditional models

- 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.

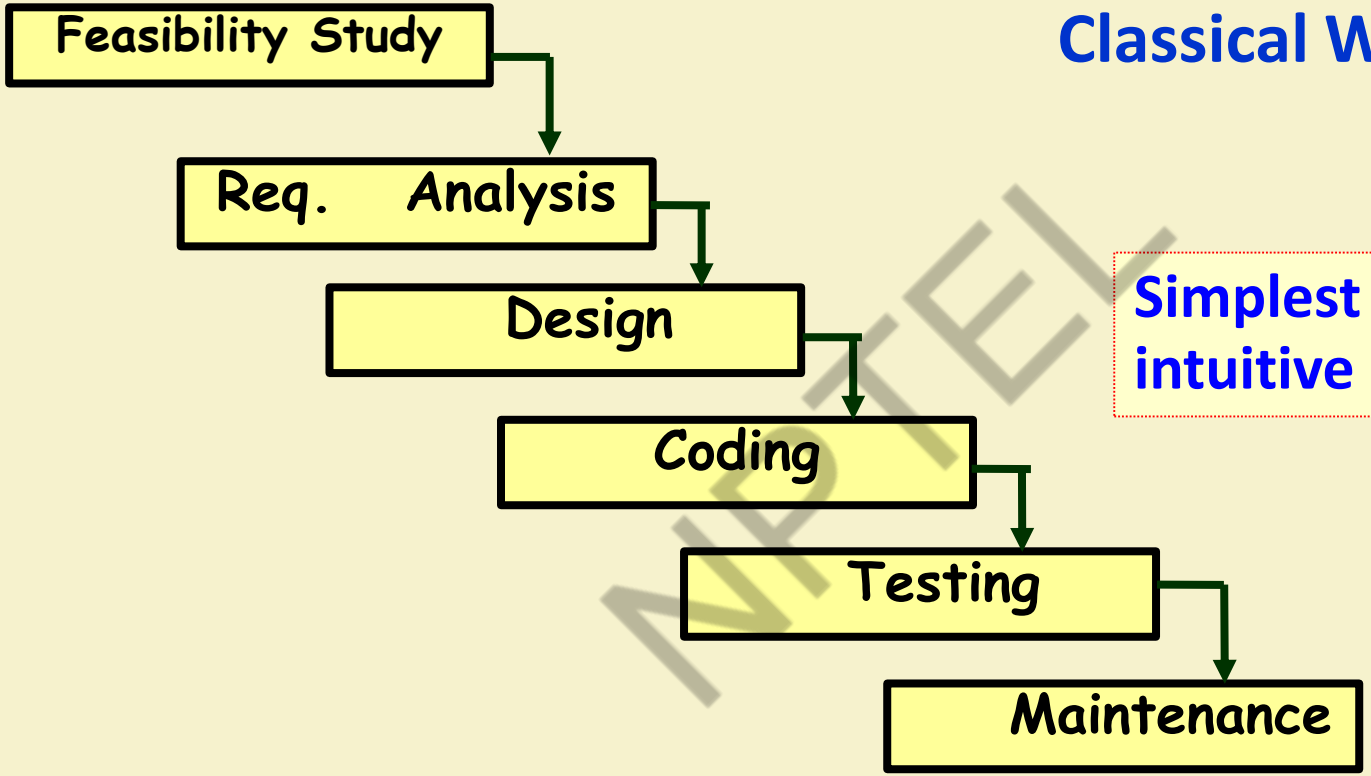
Software Life Cycle

Classical Waterfall Model

- Classical waterfall model divides life cycle into following phases:
  - Feasibility study,
  - Requirements analysis and specification,
  - Design,
  - Coding and unit testing,
  - Integration and system testing,
  - Maintenance.



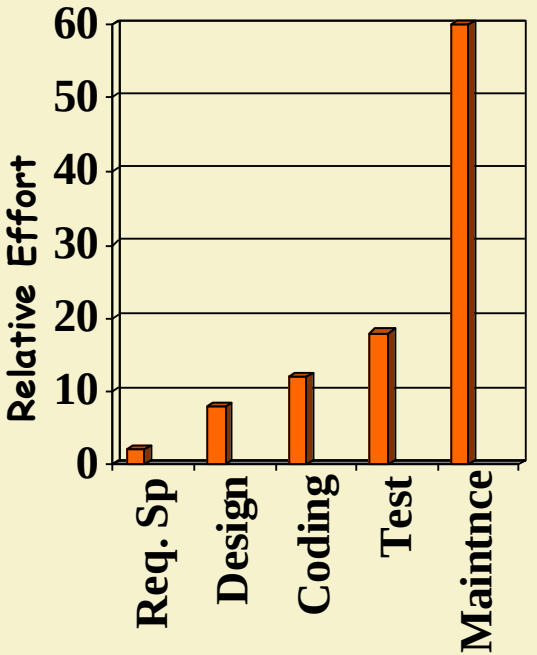
Classical Waterfall Model



Simplest and most intuitive

Relative Effort for Phases

- Phases between feasibility study and testing
  - Called development phases.
- Among all life cycle phases
  - Maintenance phase consumes maximum effort.
- Among development phases,
  - Testing phase consumes the maximum effort.



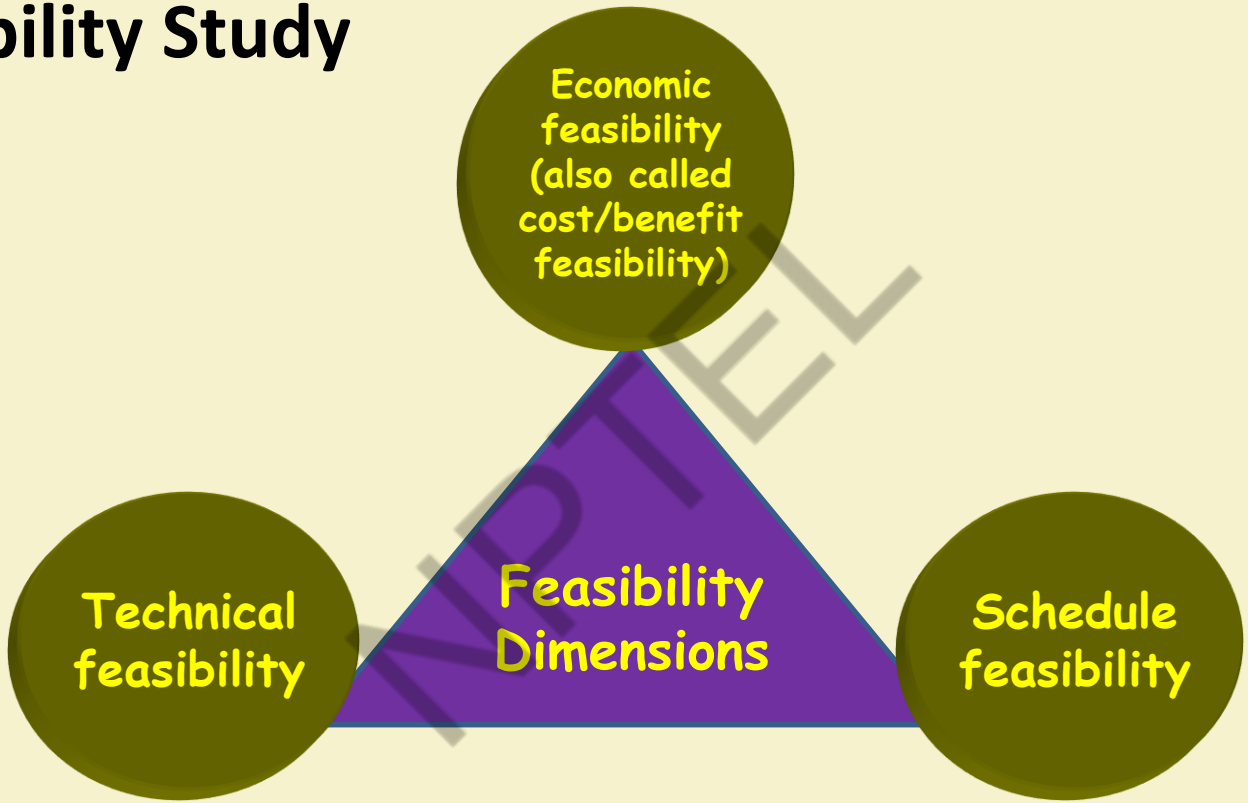
- Most organizations usually define:
  - Standards on the outputs (deliverables) produced at the end of every phase
  - Entry and exit criteria for every phase.
- They also prescribe methodologies for:
  - Specification,
  - Design,
  - Testing,
  - Project management, etc.

Process Model

Classical Waterfall Model (CONT.)

- The guidelines and methodologies of an organization:
  - –Called the organization's software development methodology.
- Software development organizations:
  - – Expect fresh engineers to master the organization's software development methodology.

Feasibility Study



- Main aim of feasibility study: determine whether developing the software is:
  - **Financially worthwhile**
  - **Technically feasible.**

Feasibility Study

- Roughly understand what customer wants:
  - **Data which would be input to the system,**
  - **Processing needed on these data,**
  - **Output data to be produced by the system,**
  - **Various constraints on the behavior of the system.**

First Step

- SPF Scheme for CFL
- CFL has a large number of employees, exceeding 50,000.
- Majority of these are casual labourers
- Mining being a risky profession:
  - Casualties are high
- Though there is a PF:
  - But settlement time is high
- There is a need of SPF:
  - For faster disbursement of benefits

Case Study

Feasibility: Case Study

- Manager visits main office, finds out the main functionalities required
- Visits mine site, finds out the data to be input
- Suggests alternate solutions
- Determines the best solution
- Presents to the CFL Officials
- Go/No-Go Decision

Activities During Feasibility Study

- Work out an overall understanding of the problem.
- Formulate different solution strategies.
- Examine alternate solution strategies in terms of:
  - resources required,
  - cost of development, and
  - development time.

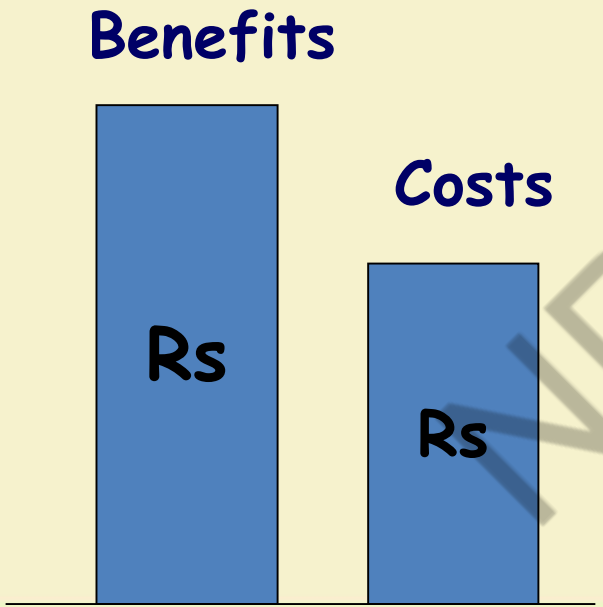
- Perform a cost/benefit analysis:
  - Determine which solution is the best.
  - May also find that none of the solutions is feasible due to:
    - high cost,
    - resource constraints,
    - technical reasons.

Activities during Feasibility Study

Cost benefit analysis (CBA)

- Need to identify all costs --- these could be:
  - **Development costs**
  - **Set-up**
  - **Operational costs**
- Identify the value of benefits
- Check benefits are greater than costs

The business case



- Benefits of delivered project must outweigh costs
- Costs include:
  - Development
  - Operation
- Benefits:
  - Quantifiable
  - Non-quantifiable

Thank You!!