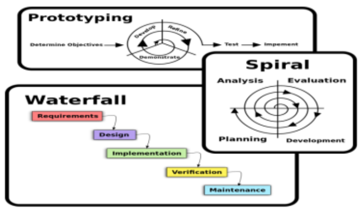
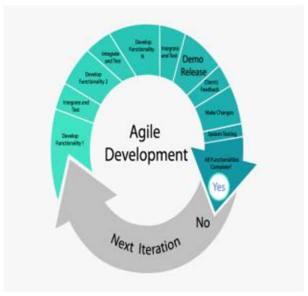
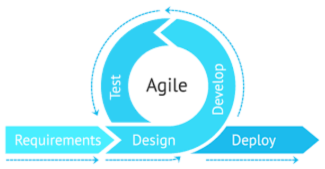


Software Engineering And Testing

Introduction





Outline

- Evolution an Engineering Discipline
- A solution to the Software Crisis
- Software Development Projects
  - Types of Software Development Projects
- Exploratory Style of Software Development
- Perceived Problem Complexity
- Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations
- Emergence of Software Engineering
  - Early Computer Programming
  - High-level Language Programming
  - Control Flow-based Design
  - Data Structure- oriented Design

Outline

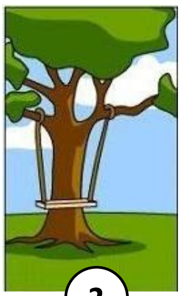
- ○ Data Flow-oriented Design
- ○ Object-oriented Design

○ Notable Changes in Software Development Practices.

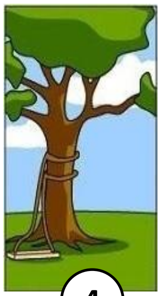
Why to Study Software Engineering?

○ Software Development Life Cycle without Software Engineering

1 How the Customer Explains Requirement


2 How the Project Leader understand it

3 How the System Analyst design it

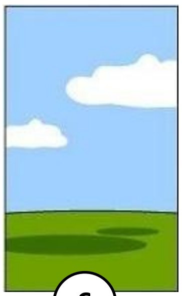

4 How the Programmer Works on it

Why to Study Software Engineering?

- Software Development Life Cycle without Software Engineering

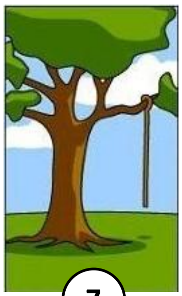
5

How the Business Consultant describe it



6

How the Project documented



7

What Operations Installed

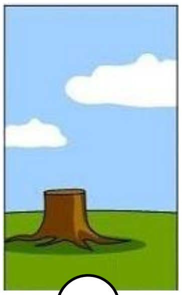
5

Why to Study Software Engineering?

- Software Development Life Cycle without Software Engineering

8

How the Customer billed



9

How it was supported

10

What the customer really needed

6

SDLC without Software Engineering

Customer Requirement	Solution
• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous	• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous
	Our value added, also gives milk

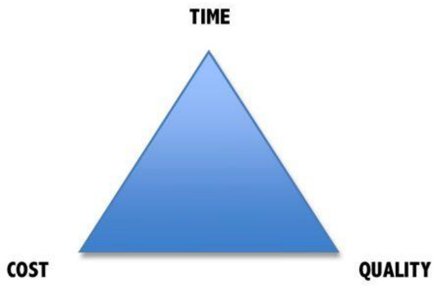
The software development **process** needs to be **engineered** to avoid the **communication gap** & to **meet the actual requirements** of customer within **stipulated budget** & **time**

Software Engineer

- Software engineers build and support software, and virtually everyone in the industrialized world uses it either directly or indirectly.
- Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high-quality computer software.

Why is Software Engineering important?

- Complex systems need a disciplined approach for designing, developing and managing them.

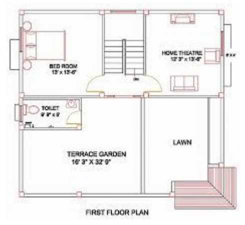


- “you can have it fast, you can have it right, or you can have it cheap. “
- Pick two!”

9

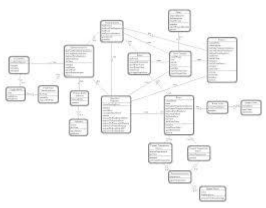
Software Engineer

Engineering



Design	Build	Product

Software Engineering



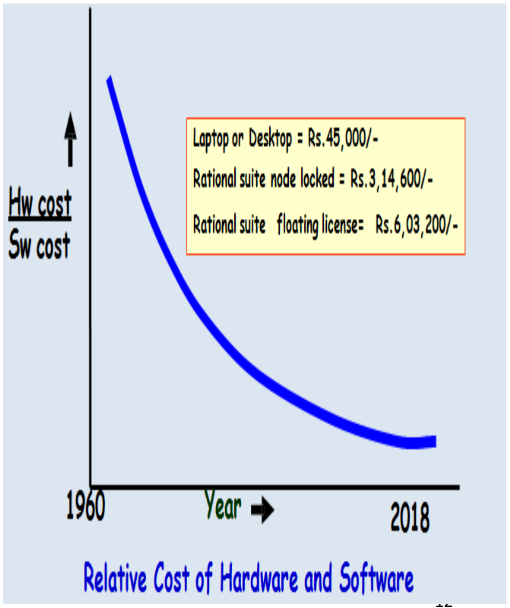
10

Software Development Crises

- Projects were:
  - Late.
  - Over budget.
  - Unreliable.
  - Difficult to maintain.
  - Performed poorly.
- Software errors….the cost
  - Errors in computer software can have devastating effects.

Software Crises

- Software products:
  - Fail to meet user requirements.
  - Frequently crash.
  - Expensive.
  - Difficult to alter, debug, and enhance.
  - Often delivered late.
  - Use resources non-optimally.



Relative Cost of Hardware and Software

Software Crises

- Example 1: 2009,Computer glitch delays flights
- Saturday 3rd October 2009-London, England (CNN)
- Dozens of flights from the UK were delayed Saturday after a glitch in an air traffic control system in Scotland, but the problem was fixed a few hours later.
- The agency said it reverted to backup equipment as engineering worked on the system.
- The problem did not create a safety issue but could cause delays in flights.

13

Software Crises

- Example 2: Ariane 5 Explosion
- European Space Agency spent 10 years and $7 billion to produce Ariane 5.
- Crash after 36.7 seconds.
- Caused by an overflow error. Trying to store a 64-bit number into a 16-bit space.
- Watch the video: http://www.youtube.com/watch?v=z-r9cYp3tTE

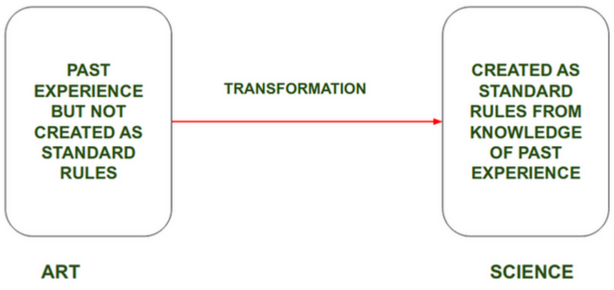
14

Software Crises

- Example 3: 1992, London Ambulance Service
- Considered the largest ambulance service in the world.
- Overloaded problem.
- It was unable to keep track of the ambulances and their statuses. Sending multiple units to some locations and no units to other locations.
- Generates many exceptions messages.
- 46 deaths.

15

Software Engineering Science Vs Art

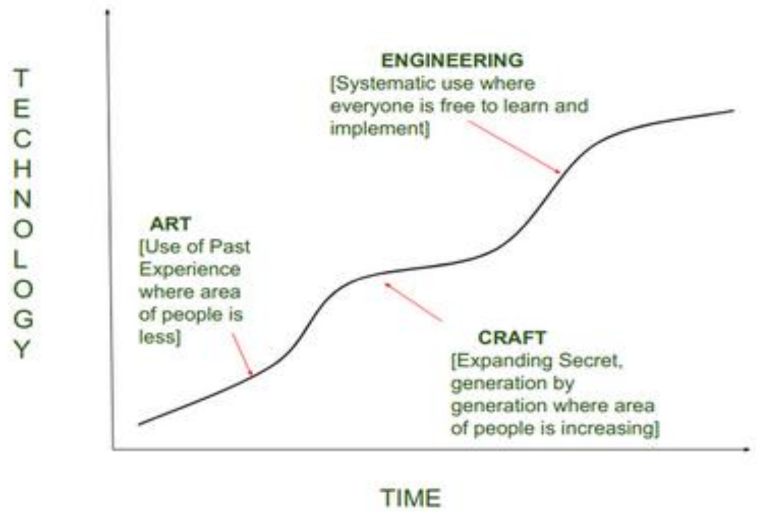

PAST EXPERIENCE BUT NOT CREATED AS STANDARD RULES
TRANSFORMATION
CREATED AS STANDARD RULES FROM KNOWLEDGE OF PAST EXPERIENCE
ART
SCIENCE

Software Engineering

- Therefore…
  - A well-disciplined approach to software development and management is necessary. This is called engineering.
- The term software engineering first appeared in the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding what was then called the “software crisis”
- “An engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.”
- Engineering approach to develop software. Systematic collection of past experience: Techniques, Methodologies, Guidelines.

17

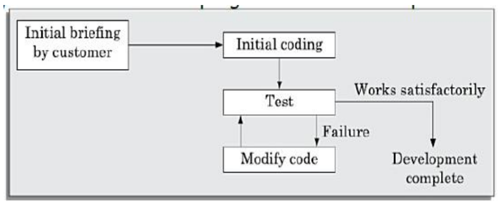
Technology Development Pattern

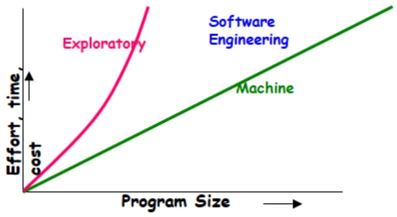


18

Evolution of an Art into an Engineering Discipline

- The early programmers used an exploratory (also called build and fix) style.
  - In the build and fix (exploratory) style, normally a `dirty' program is quickly developed.
  - The different imperfections that are subsequently noticed are fixed.
- What is Wrong with the Exploratory Style? Can successfully be used for very small programs only.





Evolution of an Art into an Engineering Discipline

- What is Wrong with the Exploratory Style?
  - Can successfully be used for very small programs only.
  - 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 a team development environment.

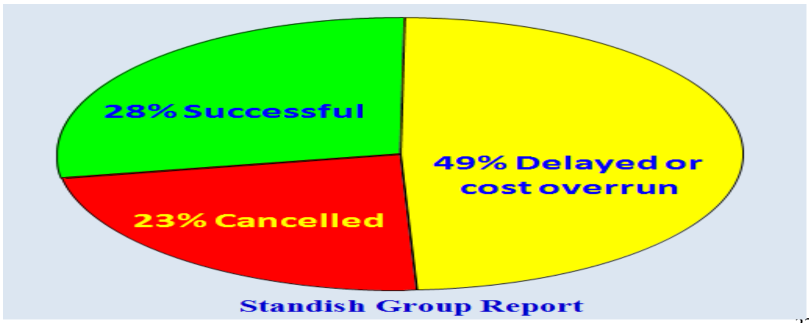
Software

- Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product.
  - As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware (can vary from bit to tons of bits)
  - As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).
- Software delivers the most important product of our time—information.

21

Software

- Vast increases in memory and storage capacity, and a wide variety of exotic input and output options, have all precipitated more complex computer-based systems.
- Complexity can produce dazzling results when a system succeeds, but they can also pose huge problems for those who must build complex systems.



22

Questions Asked

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

23

Software is dead…..!

- **The old School view of Software**
  - You buy it
  - You own it &
  - It's your job to manage it
  - That is coming to an end
- **Because of web 2.0 & extensive computing power, there is a different generation of software**
  - It is delivered via Internet
  - It looks exactly like it's residing on each user's computing device
  - Actually it reside on far away server

24

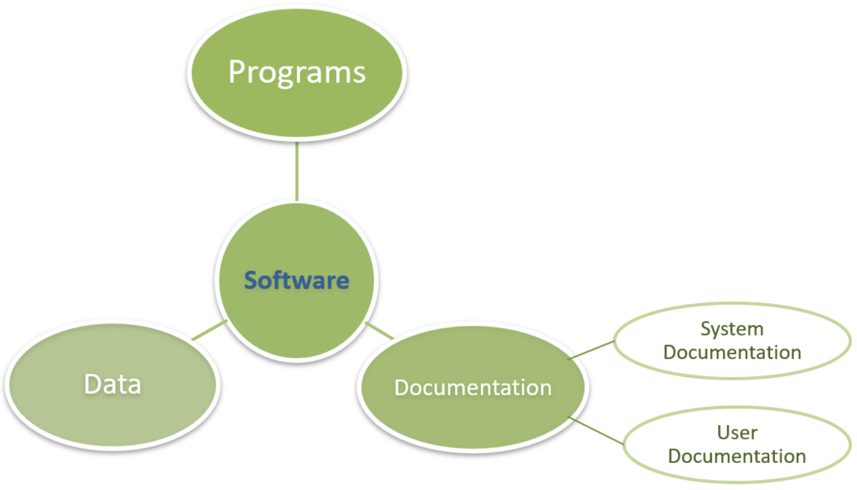
Types of Software

- Generic products.
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.

Types of Software

- Customized products.
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.
- Outsourced projects

What is Software?



Program vs. Software?

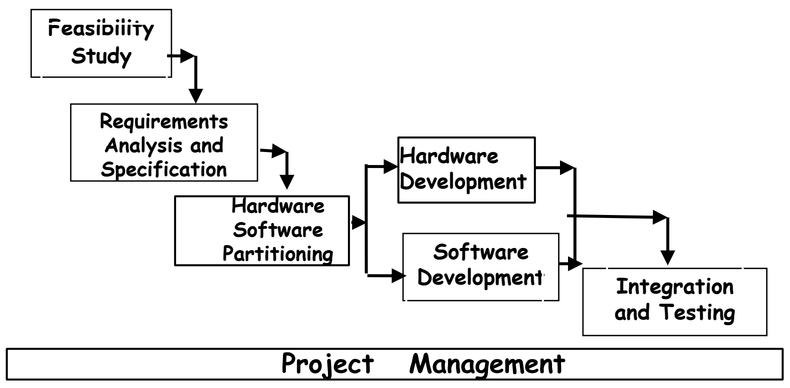
Program	Software
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development.	Systematic development
Programmer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

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.
- The high-level problem:
  - Deciding which tasks are to be solved by software.
  - Which ones by hardware
- Often, hardware and software are developed together: Hardware simulator is used during software development.
- Final system testing

29

Computer Systems Engineering

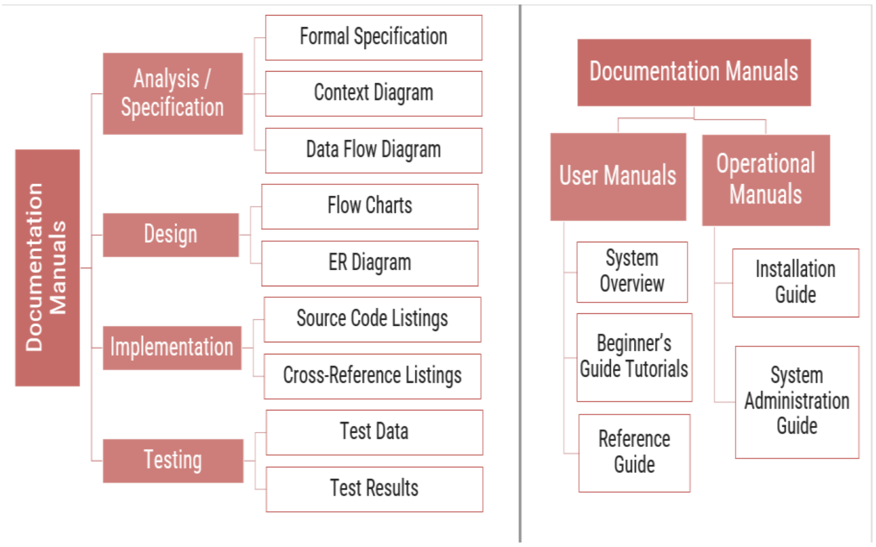


Defining Software

- Software is:
  - instructions (computer programs) that when executed provide desired features, function, and performance;
  - data structures that enable the programs to adequately manipulate information and
  - documentation that describes the operation and use of the programs.
- Software is a logical rather than a physical system element.

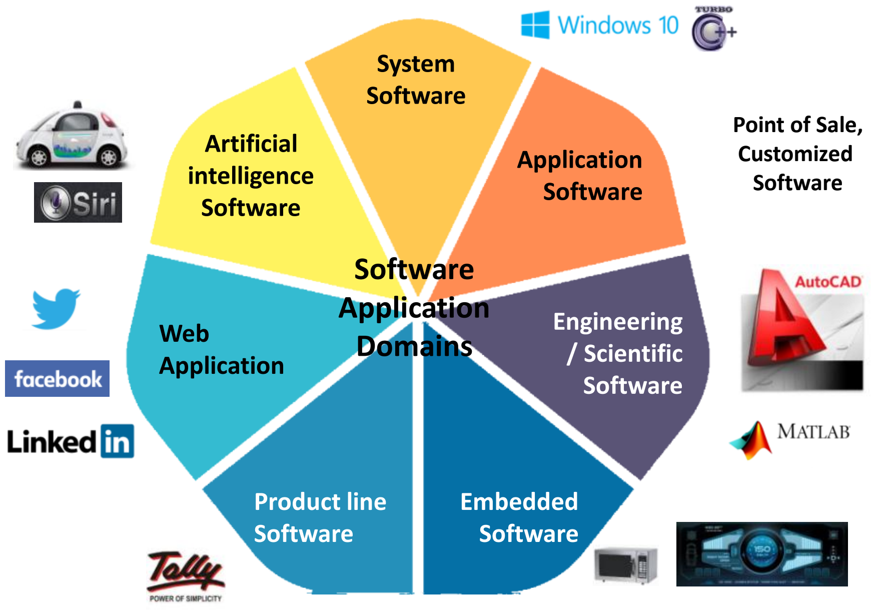
31

List of documentation & manuals

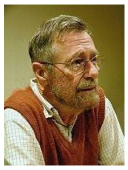


32

Software Applications Domains



Software Engineering vs. Computer Science



- "Computer science is no more about computers than astronomy is about telescopes." Edsger Dijkstra

Computer Science	Software Engineering
• Theory. • Fundamentals.	• Practicalities of software design, development and delivery.

34

Software Engineering vs. Systems Engineering

- Systems Engineering:
  - Interdisciplinary engineering field (computer, software, and process eng.).
  - Focuses on how complex engineering projects should be designed and managed.

Systems Engineering	Software Engineering
• All aspects of computer-based systems development: HW + SW + Process. • Older than SWE.	• Deals with the design, development and delivery of SW. • Is part of Systems Engineering.

35

Software Engineering

- Some realities:
  - a concerted effort should be made to understand the problem before a software solution is developed
  - design becomes a pivotal activity
  - software should exhibit high quality
  - software should be maintainable
- These simple realities lead to one conclusion: software in all of its forms and across all of its application domains should be engineered.

36

Definition

- 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.- Fritz Bauer
- it omits mention of the importance of measurement and metrics; it does not state the importance of an effective process. And yet, Bauer's definition provides us with a baseline.
- The IEEE definition: 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.

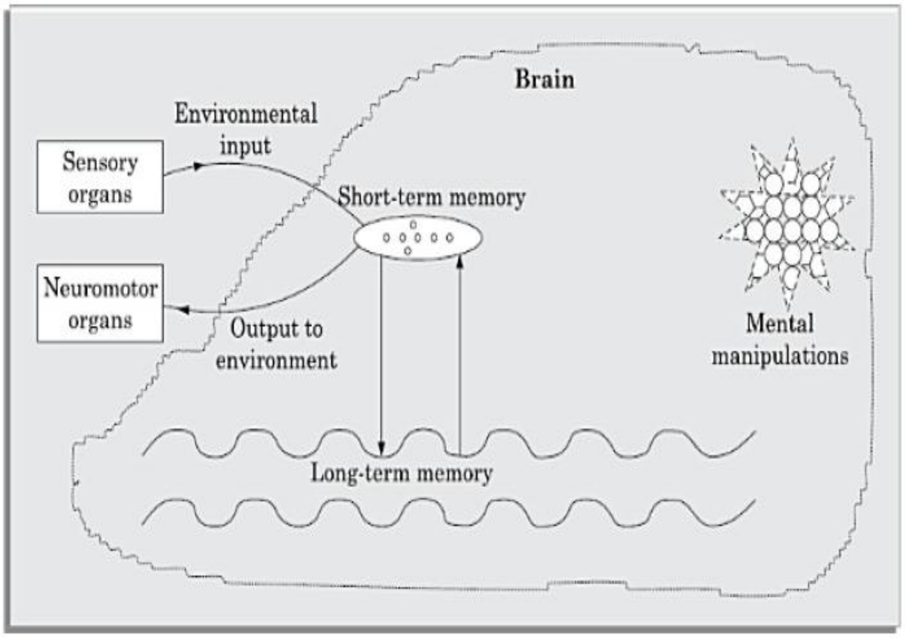
  (2) The study of approaches as in (1).

37

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.
- If you are asked the question: ``If it is 10AM now, how many hours are remaining today?"
  - First, 10AM would be stored in the short-term memory.
  - Next, a day is 24 hours long would be fetched from the long term memory into short term memory.
  - Finally, the mental manipulation unit would compute the difference (24-10).

Human Cognition Mechanism



Short term Memory

- An item stored in the short term memory can get lost:
  - Either due to decay with time or
  - Displacement by newer information.
- This restricts the time for which an item is stored in short term memory to few tens of seconds.
- However, an item can be retained longer in the short term memory by recycling.

Long term Memory

- The size of the long term memory can vary from several million items to several billion items, largely depending on how actively a person exercises his mental faculty.
- is usually retained for several years.
- Items present in the short-term memory can get stored in the long-term memory either through large number of refreshments (repetitions) or by forming links with already existing items in the long-term memory.

What is an item

- An item is any set of related information.
  - A character such as `a' or a digit such as `5' can be items.
  - A word, a sentence, a story, or even a picture can each be a single item.
- 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 can be stored using only one place in the memory.
- If you are given the binary number 110010101001
- It may prove very hard for you to understand and remember.
- But, the octal form of 6251 (i.e. (110)(010)(101)(001)) would be easier.
- You have managed to create chunks of three items each.

Evidence of Short Term Memory

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

43

The Magical Number 7

- If a person deals with seven or less number items: (7± 2)
  - These would be easily be accommodated in the short term memory.
  - So, he can easily understand it.
- As the number of new information increases beyond seven,

  It becomes exceedingly difficult to understand it.

44

Implication in Program Development

- A small program having just a few variables:
  - Is within the 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.
- Instead of a human, if a machine could be writing (generating) a program, The slope of the curve would be linear.
- But, why does the effort-size curve become almost linear when software engineering principles are deployed?
  - Software engineering principles extensively use techniques specifically to overcome the human cognitive limitations.

Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations

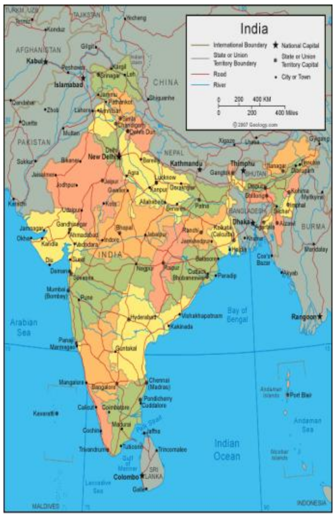
- Mainly two important principles are deployed:
  - Abstraction
  - Decomposition

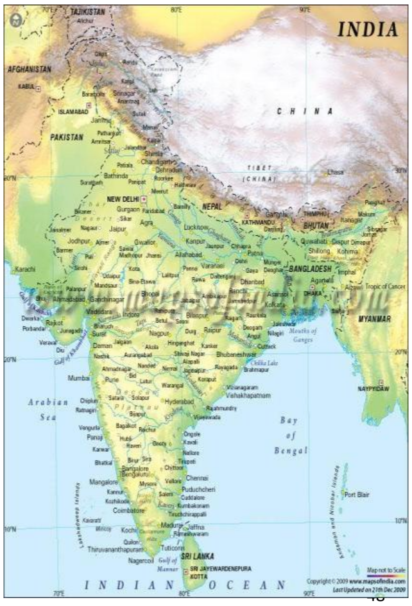
Abstraction

- Simplify a problem by omitting unnecessary details.
  - Focus attention on only one aspect of the problem and ignore irrelevant details.
  - Also called model building.
- Suppose you are asked to develop an overall understanding of some country.
  - 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.
- A map, in fact, is an abstract representation of a country.

47

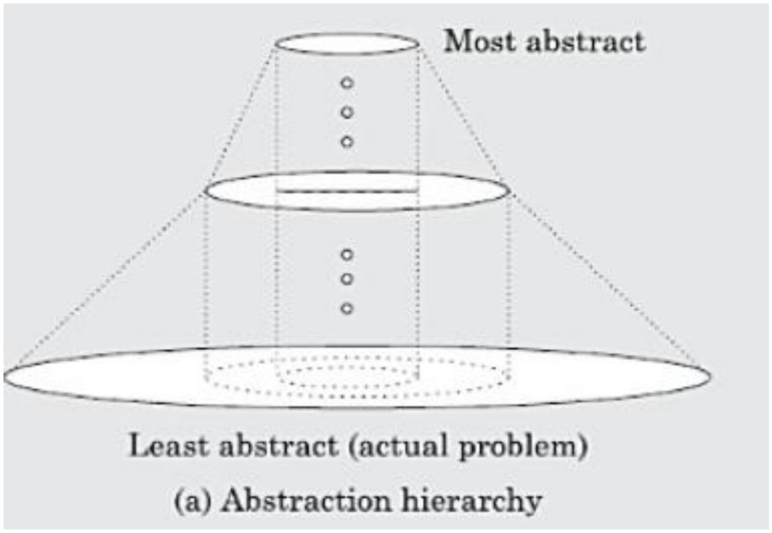
You would study Abstraction





48

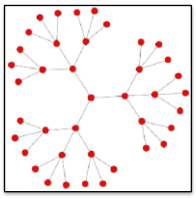
Abstraction



(a) Abstraction hierarchy

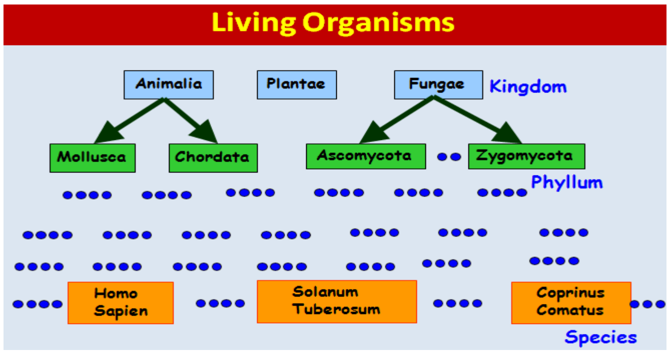
Abstractions of Complex Problems

- **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.



Abstractions of Complex Problems [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.**



51

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 can be easily solved.
  - The full problem is solved when all the parts are solved.

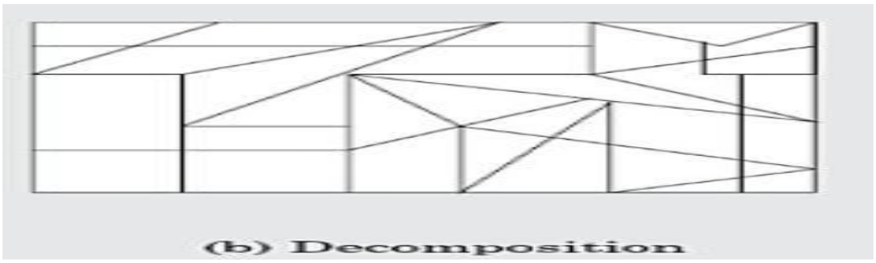
- A popular way to demonstrate the decomposition principle:
  - Try to break a bunch of sticks tied together versus breaking them individually.

- 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.

52

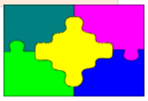
Decomposition

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



53

Why Study Software Engineering?

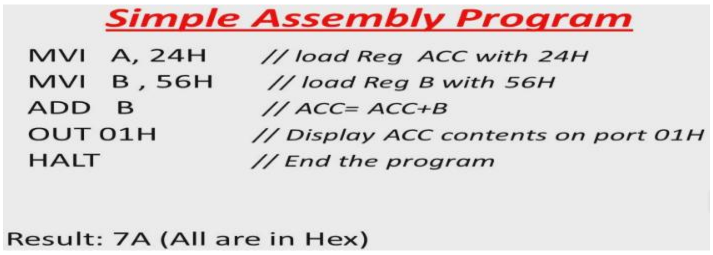


- **To acquire skills to develop large programs.**
  - Handling exponential growth in complexity with size.
  - Systematic techniques based on abstraction (modelling) and decomposition.
- **Learn systematic techniques of:**
  - Specification, design, user interface development, testing, project management, maintenance, etc.
  - Appreciate issues that arise in team development.
- **To acquire skills to be a better programmer:**
  - Higher Productivity
  - Better Quality Programs

54

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.



Simple Assembly Program

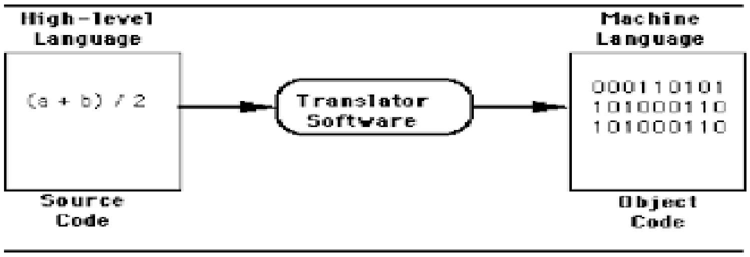
```
MVI   A, 24H      // load Reg  ACC with 24H
MVI   B , 56H      // load Reg B with 56H
ADD   B           // ACC= ACC+B
OUT 01H           // Display ACC contents on port 01H
HALT              // End the program

Result: 7A (All are in Hex)
```

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

Emergence of Software Engineering Techniques

- **High-Level Language Programming (Early 60s):**
- **High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:**
  - This reduced software development efforts greatly.
- **Software development style was still exploratory.**
  - Typical program sizes were limited to a few thousands of lines of source code.



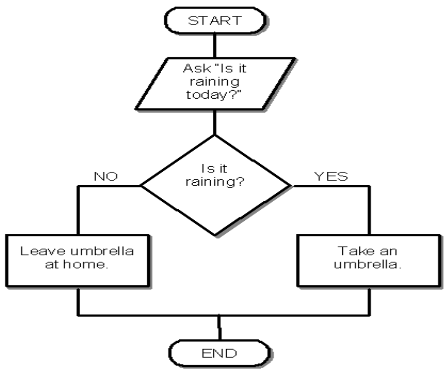
Emergence of Software Engineering Techniques

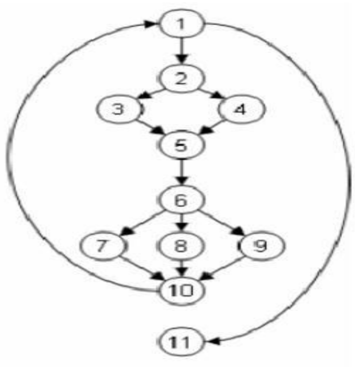
- **Control Flow-Based Design (late 60s):**
- **Size and complexity of programs increased further:**
  - Exploratory programming style proved to be insufficient.
- **Programmers found:**
  - Very difficult to write cost-effective and correct programs.
- **Programmers found it very difficult:**
  - To 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."**

57

Emergence of Software Engineering Techniques

- **Control Flow-Based Design (late 60s):**
- **What is a program's control structure?**
  - Sequence in which a program's instructions are executed.
- **To help design programs having good control structure:**
  - Flow charting technique was developed.

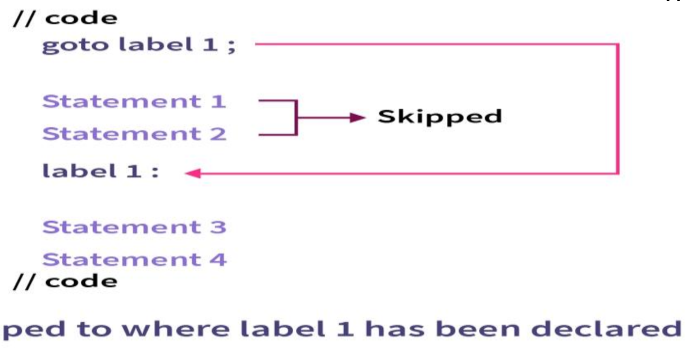




3

Emergence of Software Engineering Techniques

- **Control Flow-Based Design (late 60s):**
- **What causes program complexity?**
  - GOTO statements makes control structure of a program messy.
  - GOTO statements alter the flow of control arbitrarily.
  - The need to restrict use of GOTO statements was recognized.



```
#include <stdio.h>
int main(){    // we will print numbers from start to end
  // initialize start and end variables
  int start = 1, end = 10;
  // initialize variable to keep track of which number is to be printed
   int curr = start;
  // defining the label
   print_line :
  // print the current number
   printf("%d ", curr);
  // check if the current has reached the end if not, that means some numbers are still to be printed
  if(curr<end)      {      // increment current
        curr++;
       // use goto to again repeat
        goto print_line;       }
   // if the current has reached the end, the statements inside if will not be executed the program terminates
  return 0;}
```

```
1 2 3 4 5 6 7 8 9 10
```

Emergence of Software Engineering Techniques

- **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 GOTO 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
```

61

Emergence of Software Engineering Techniques

- **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.**
- **Some programmers published several counter articles:**
  - Highlighted the advantages and inevitability of GOTO statements.
- **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;)

62

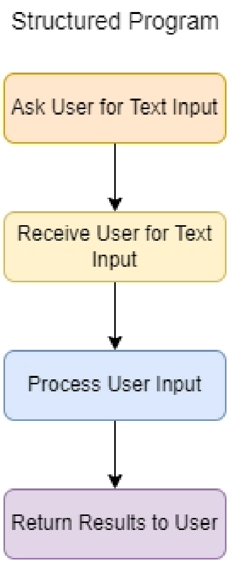
Emergence of Software Engineering Techniques

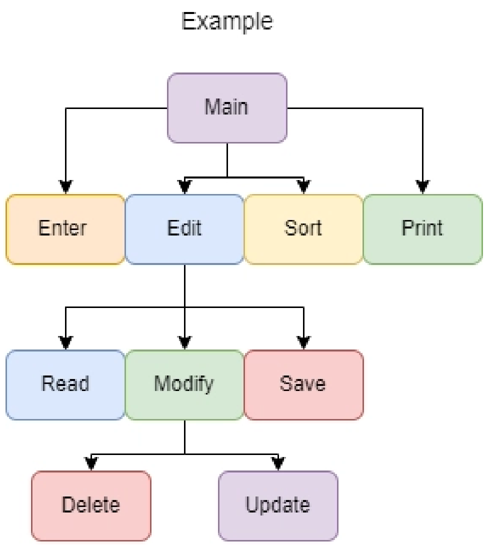
- **Control Flow-Based Design (late 60s):**
- **Everyone accepted:**
  - It is possible to solve any programming problem without using GOTO statements.
  - This formed the basis of Structured Programming methodology.

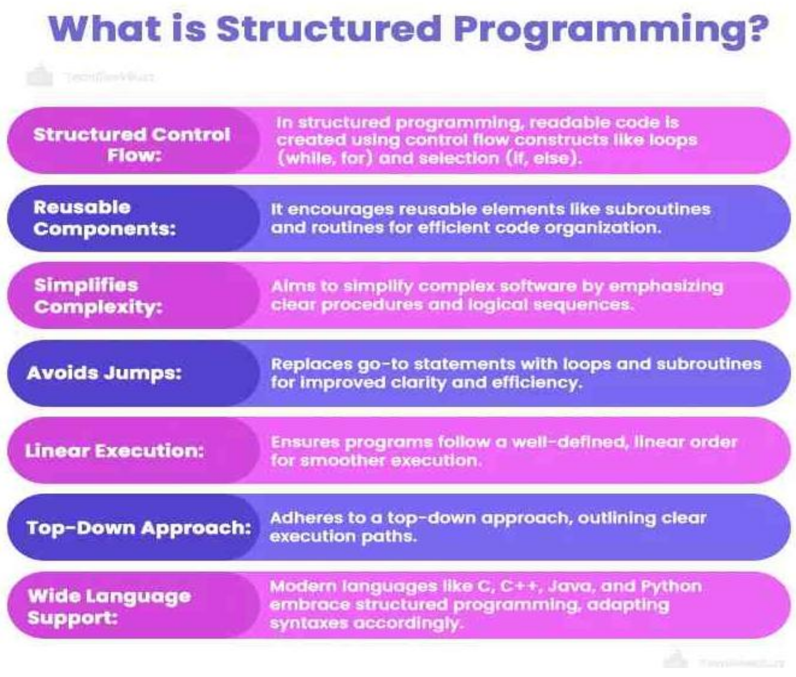
Emergence of Software Engineering Techniques

- **Structured Programming:**
- **A program is called structured:**
  - When it uses only the following types of constructs:
    - Sequence,
    - Selection,
    - Iteration
  - Consists of modules.
- **Sometimes, violations to structured programming are permitted:**
  - Due to practical considerations such as: Premature loop exit (break) or for exception handling.

Emergence of Software Engineering Techniques







What is Structured Programming?

Structured Control Flow: In structured programming, readable code is created using control flow constructs like loops (while, for) and selection (if, else).

Reusable Components: It encourages reusable elements like subroutines and routines for efficient code organization.

Simplifies Complexity: Aims to simplify complex software by emphasizing clear procedures and logical sequences.

Avoids Jumps: Replaces go-to statements with loops and subroutines for improved clarity and efficiency.

Linear Execution: Ensures programs follow a well-defined, linear order for smoother execution.

Top-Down Approach: Adheres to a top-down approach, outlining clear execution paths.

Wide Language Support: Modern languages like C, C++, Java, and Python embrace structured programming, adapting syntaxes accordingly.

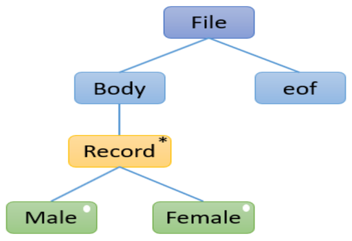
Emergence of Software Engineering Techniques

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

67

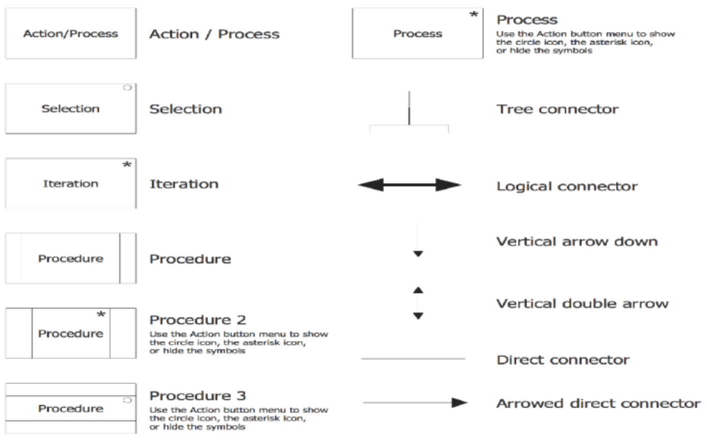
Emergence of Software Engineering Techniques

- **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.
- **An example of data structure-oriented design technique:**
  - Jackson's Structured Programming(JSP) methodology, Developed by Michael Jackson in 1970s.
  - Program code structure should correspond to the data structure.

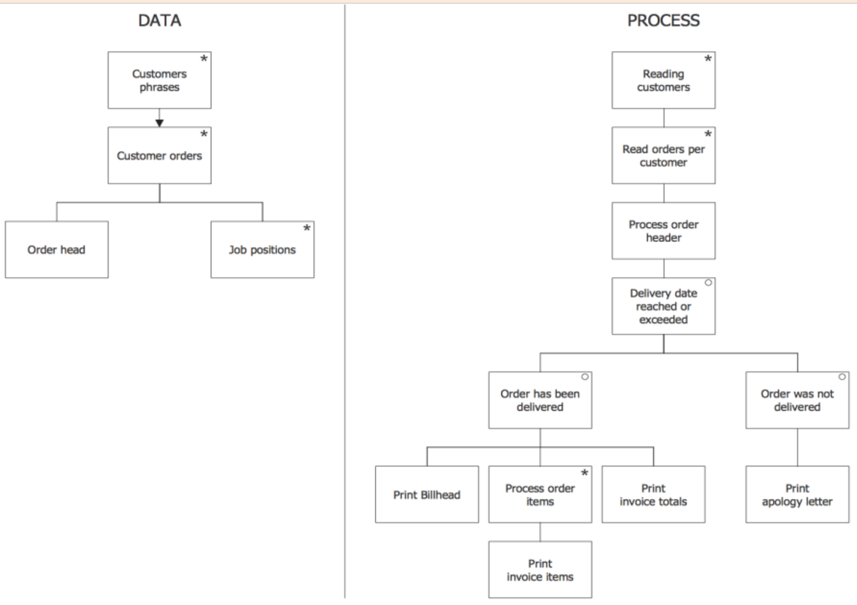


Emergence of Software Engineering Techniques

- **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.

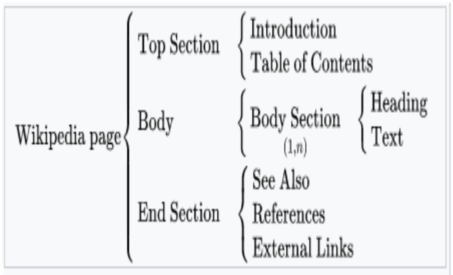

Action/Process
Action / Process
Process
Process
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Selection
Selection
Tree connector
Iteration
Iteration
Logical connector
Procedure
Procedure
Vertical arrow down
Vertical double arrow
Procedure
Procedure 2
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Direct connector
Procedure
Procedure 3
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Arrowed direct connector

Jackson Diagram Example

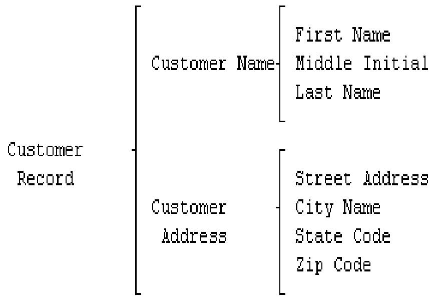


Emergence of Software Engineering Techniques

- **Data Structure-Oriented Design (Early 70s) :**
- Several other data structure-oriented Methodologies also exist:
  - e.g., Warnier-Orr Methodology.

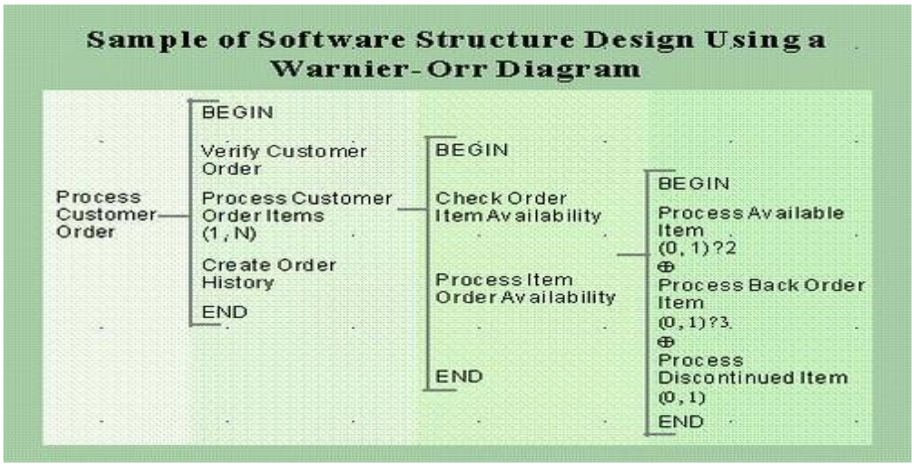


Sample Warnier Orr Data diagram illustrating structure of a Wikipedia page.



Emergence of Software Engineering Techniques

- **Data Structure-Oriented Design (Early 70s) :**
- Several other data structure-oriented Methodologies also exist:
  - e.g., Warnier-Orr Methodology.


Sample of Software Structure Design Using a Warnier-Orr Diagram
Process Customer Order
BEGIN
Verify Customer Order
Process Customer Order Items (1, N)
Create Order History
END
BEGIN
Check Order Item Availability
Process Item Order Availability
END
BEGIN
Process Available Item (0, 1)?2
⊕
Process Back Order Item (0, 1)?3
⊕
Process Discontinued Item (0, 1)
END

Emergence of Software Engineering Techniques

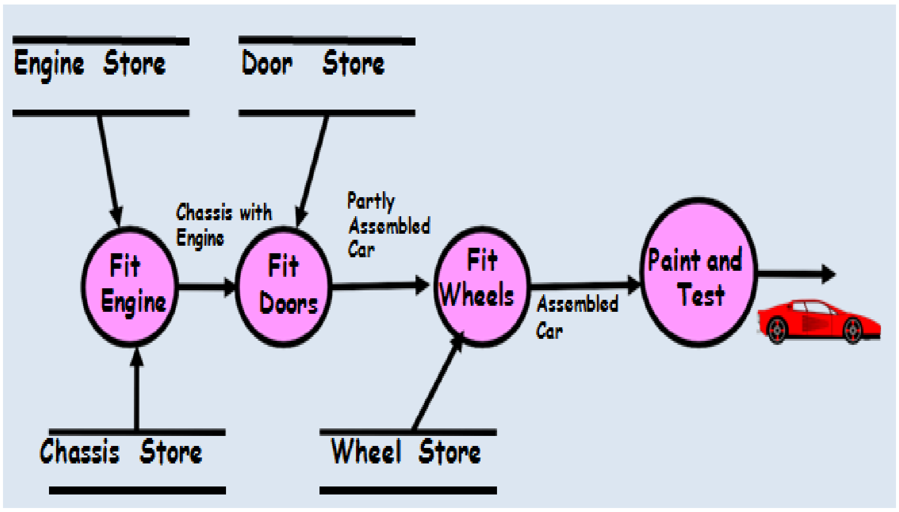
- **Data Flow-Oriented (Late 70s) :**
- **Data flow-oriented techniques advocate:**
  - The data items input to a system must first be identified,
  - Processing required on the data items to produce the required outputs should be determined.
- **Data flow technique identifies:**
  - Different processing stations (functions) in a system.
  - The items (data) that flow between processing stations.
- **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.**

73

Emergence of Software Engineering Techniques

- **Data Flow-Oriented (Late 70s) :**

Data Flow Model of a Car Assembly Unit



Emergence of Software Engineering Techniques

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

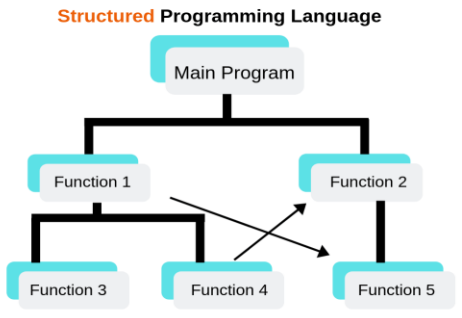
75

Emergence of Software Engineering Techniques

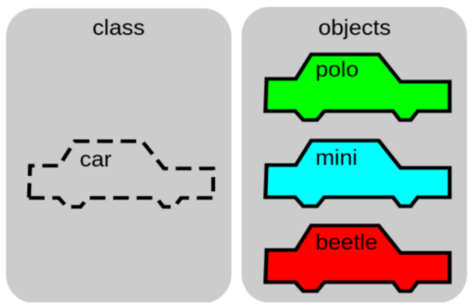
- **Object-Oriented Design (80s) :**
- **Object-oriented techniques have gained wide acceptance:**
  - Simplicity
  - Reuse possibilities
  - Lower development time and cost
  - More robust code
  - Easy maintenance

76

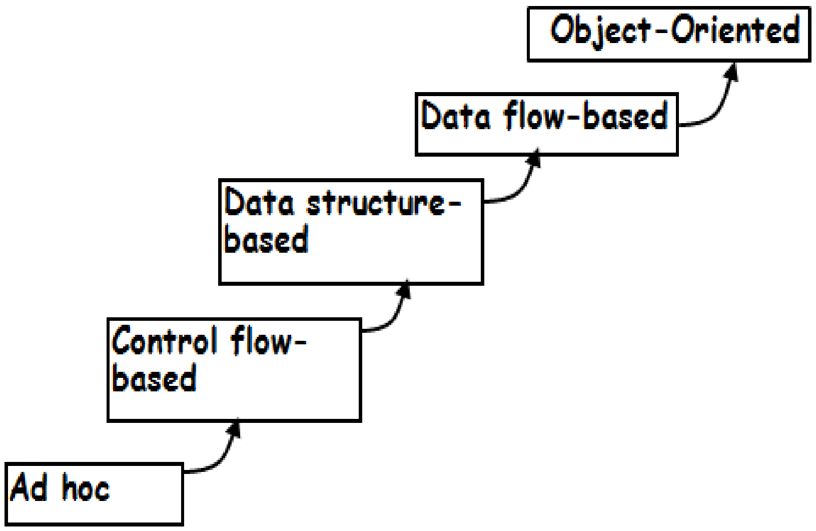
Structured Vs Object oriented



The **main difference** between structured and object oriented programming is:
★ The **structured programming** allows developing a program using a set of **modules or functions**
★ while the **object oriented programming** allows constructing a program using a set of **objects and their interactions**.



Evolution of Design Techniques



Notable Changes in S/W 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.
- In exploratory style, errors are detected only during testing,
- Now, focus is on detecting as many errors as possible in each phase of development.
- In exploratory style, coding is synonymous with program development.
- Now, coding is considered only a small part of program development effort.

79

Notable Changes in S/W Development Practices

- Differences between the exploratory style and modern software development practices:
- 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.
- During all stages of development process: Periodic reviews are being carried out .
- Software testing has become systematic: Standard testing techniques are available.

80

Notable Changes in S/W Development Practices

- Differences between the exploratory style and modern software development practices:
- 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.
- Increased visibility makes software project management easier.
- Because of good documentation: fault diagnosis and maintenance are smoother now.
- Several metrics are being used: help in software project management, quality assurance, etc.
- Projects are being thoroughly planned: estimation, scheduling, monitoring mechanisms.
- Use of CASE tools.

81

82