Software Engineering

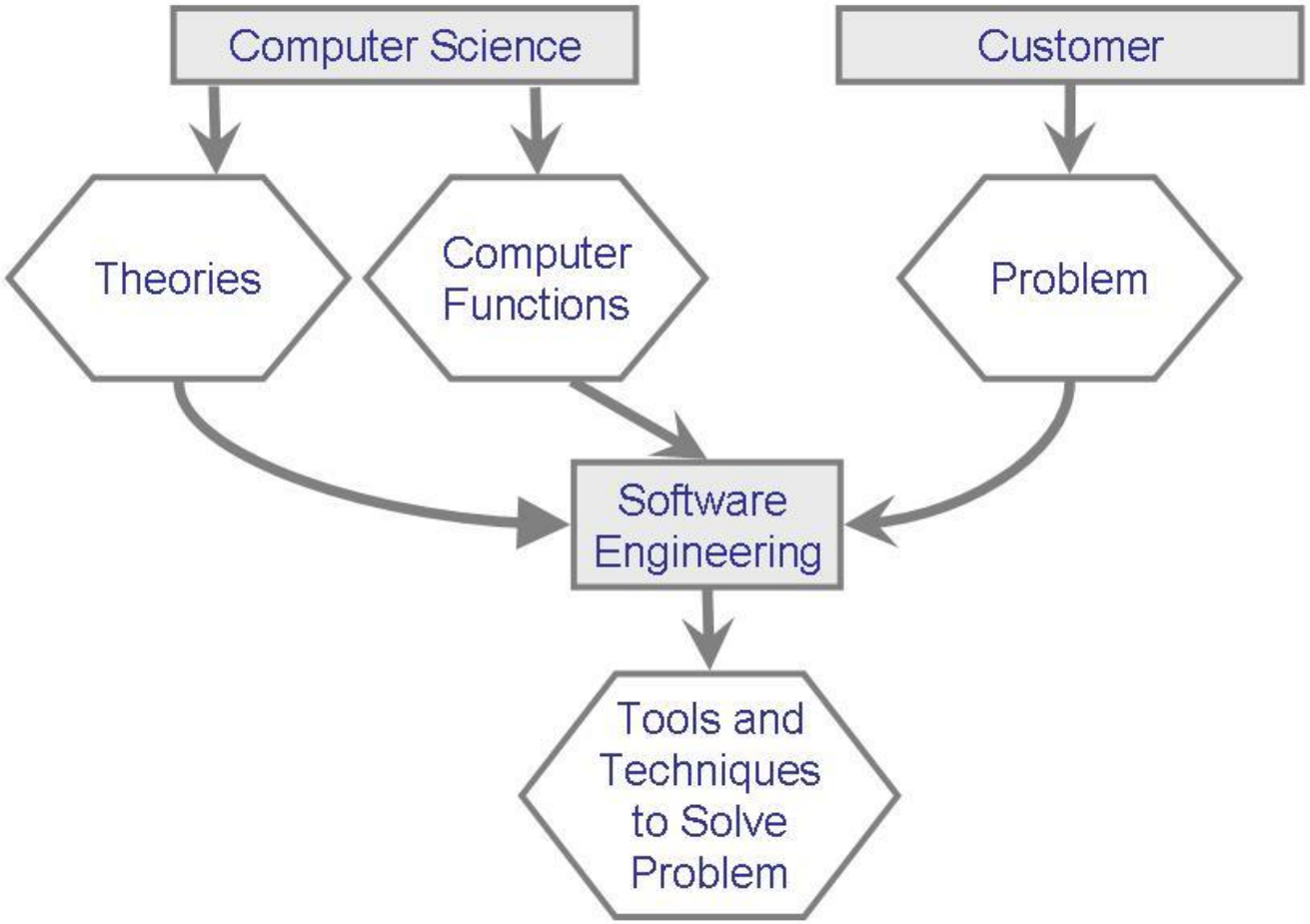
Ashish Kumar Dwivedi

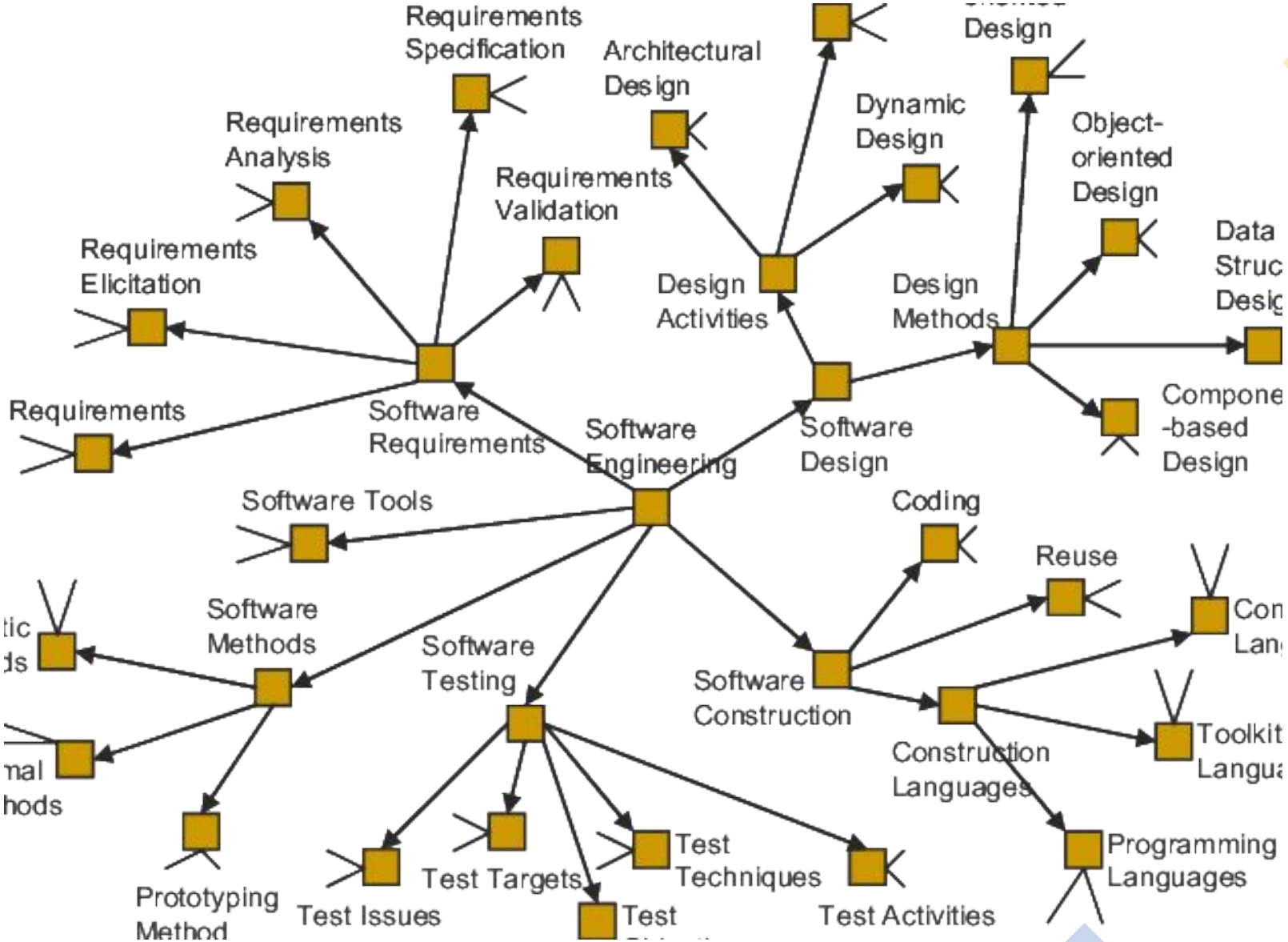
Quote:

"Ideas and technological discoveries are the driving engines of economic growth."

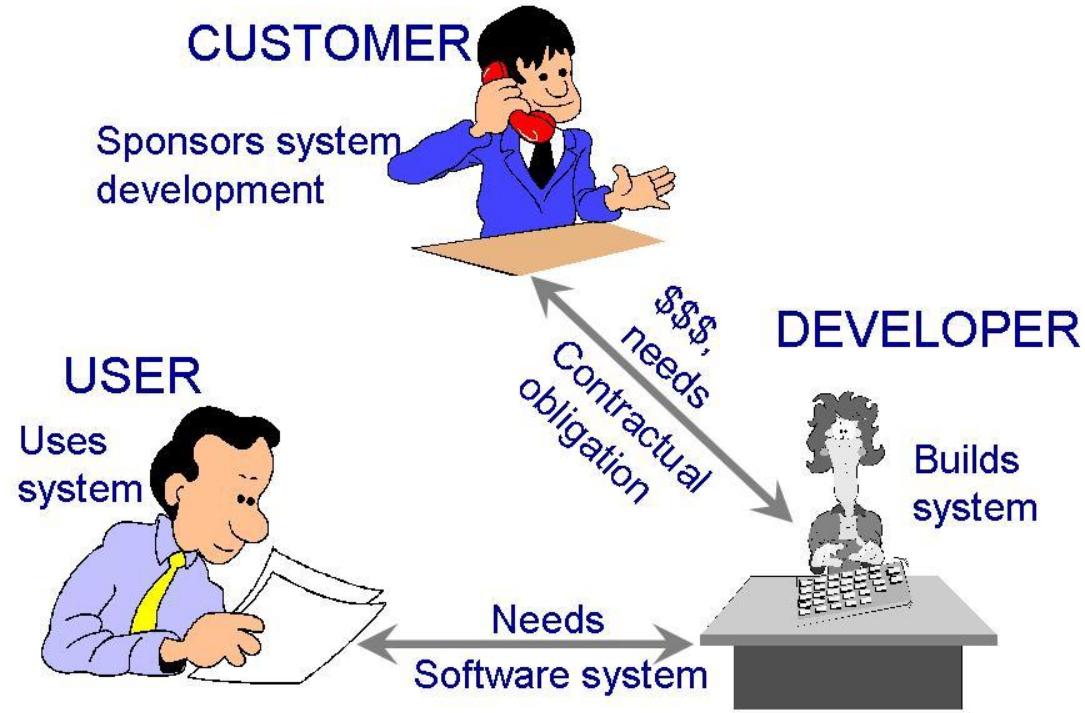
Wall Street Journal

Relationship between computer science and software engineering

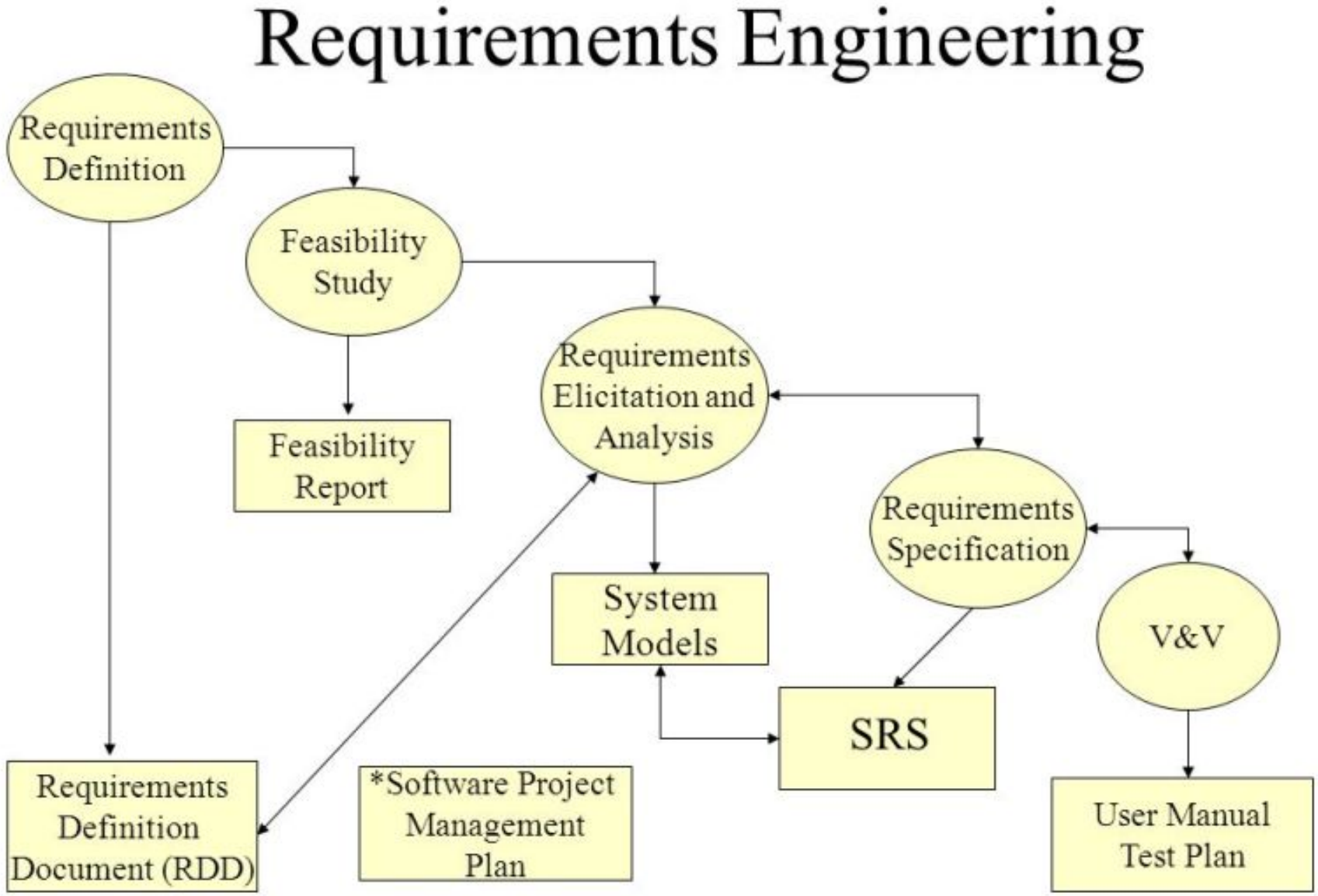



Requirements Specification
Architectural Design
Design
Requirements Analysis
Dynamic Design
Object-oriented Design
Requirements Validation
Requirements Elicitation
Design Activities
Design Methods
Data Struc Desig
Requirements
Software Requirements
Software Engineering
Software Design
Compone -based Design
Software Tools
Coding
Reuse
Software Methods
Software Testing
Software Construction
Toolkit Langua
Construction Languages
Programming Languages
Prototyping Method
Test Issues
Test Targets
Test Techniques
Test Activities

Participants in Software Development



Understanding the Requirements



Requirement Engineering Keywords

- Inception: How does a software project get started?
- Elicitation:
  - Problems of scope
  - Problems of understanding
  - Problems of volatility
- Elaboration: The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- Negotiation: Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- Specification:
- Validation:

Requirements Engineering

Objective: Requirements engineering tools assist in requirements gathering, requirements modeling, requirements management, and requirements validation.

Mechanics: Tool mechanics vary. In general, requirements engineering tools build a variety of graphical (e.g., UML) models that depict the informational, functional, and behavioral aspects of a system. These models form the basis for all other activities in the software process.

Representative Tools:[7]

A reasonably comprehensive (and up-to-date) listing of requirements engineering tools can be found at the Volvere Requirements resources site at **www.volere.co.uk/tools.htm**. Requirements modeling tools are discussed in Chapters 6 and 7. Tools noted below focus on requirement management.

EasyRM, developed by Cybernetic Intelligence GmbH (**www.easy-rm.com**), builds a project-specific dictionary/glossary that contains detailed requirements descriptions and attributes.

Rational RequisitePro, developed by Rational Software (**www-306.ibm.com/software/awdtools/reqpro/**), allows users to build a requirements database; represent relationships among requirements; and organize, prioritize, and trace requirements.

Many additional requirements management tools can be found at the Volvere site noted earlier and at **www.jiludwig.com/Requirements_Management_Tools.html**.

Design versus Coding

The scene: Jamie's cubicle, as the team prepares to translate requirements into design.

The players: Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

The conversation:

Jamie: You know, Doug [the team manager] is obsessed with design. I gotta be honest, what I really love doing is coding. Give me C++ or Java, and I'm happy.

Ed: Nah . . . you like to design.

Jamie: You're not listening; coding is where it's at.

Vinod: I think what Ed means is you don't really like coding; you like to design and express it in code. Code is the language you use to represent the design.

Jamie: And what's wrong with that?

Vinod: Level of abstraction.

Jamie: Huh?

Ed: A programming language is good for representing details like data structures and algorithms, but it's not so good for representing architecture or component-to-component collaboration . . . stuff like that.

Vinod: And a screwed-up architecture can ruin even the best code.

Jamie (thinking for a minute): So, you're saying that I can't represent architecture in code . . . that's not true.

Vinod: You can certainly imply architecture in code, but in most programming languages, it's pretty difficult to get a quick, big-picture read on architecture by examining the code.

Ed: And that's what we want before we begin coding.

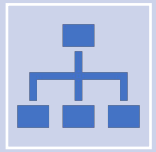
Jamie: Okay, maybe design and coding are different, but I still like coding better.

Design Principles

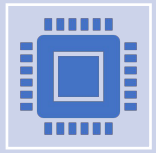
1. ***Design should be traceable to the requirements model***
2. ***Always consider the architecture of the system to be built***
3. ***Design of data is as important as design of processing functions***
4. ***Interfaces (both internal and external) must be designed with care.***
5. ***User interface design should be tuned to the needs of the end user. However, in every case, it should stress ease of use.***
6. ***Component-level design should be functionally independent.***
7. ***Components should be loosely coupled to one another and to the external environment.***
8. ***Design representations (models) should be easily understandable.***
9. ***The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.***

Design Concepts

Software design encompasses the set of principles, concepts, and practices that lead to the development of a high-quality system or product.



Design principles establish an overriding philosophy that guides you in the design work you must perform.

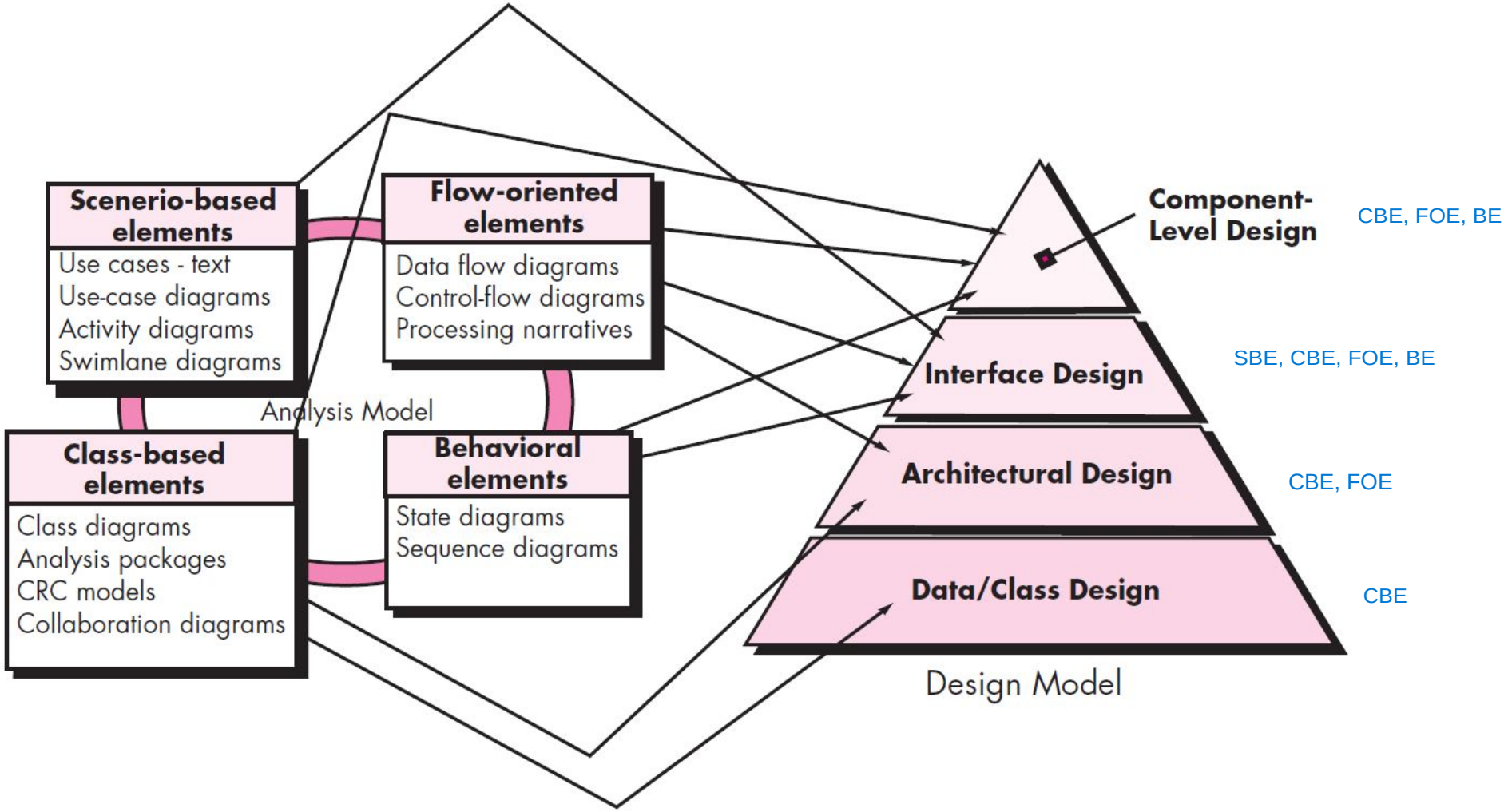


Design concepts must be understood before the mechanics of design practice are applied, and design practice itself leads to the creation of various representations of the software that serve as a guide for the construction activity that follows.

Design Fundamentals

- **Design** creates a representation or model of the software, but unlike the requirements model (that focuses on describing required data, function, and behavior), the design model provides detail about software architecture, data structures, interfaces, and components that are necessary to implement the system.
- **Why is it important?** Design allows you to model the system or product that is to be built.
- **What are the steps?** Design depicts the software in many different ways:
  - First the architecture of the system or product must be represented.
  - Then, the interfaces that connect the software to end users, to other systems and devices, and to its own constituent components are modeled.
  - Finally, the software components that are used to construct the system are designed. Each of these views represents a different design action, but all must conform to a set of basic design concepts that guide software design work.
- **What is the work product?** A design model that encompasses architectural, interface, component level, and deployment representations is the primary work product that is produced during software design.

Translating the requirements model into the design model



Design Documents

- The data/class design transforms class models into design class realizations and the requisite data structures required to implement the software.
- The objects and relationships defined in the CRC diagram and the detailed data content depicted by class attributes and other notation provide the basis for the data design action.
- The architectural design defines the relationship between major structural elements of the software, the architectural styles and design patterns that can be used to achieve the requirements defined for the system.
- The interface design describes how the software communicates with systems that interoperate with it, and with humans who use it.
- An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, usage scenarios and behavioral models provide much of the information required for interface design.
- The component-level design transforms structural elements of the software architecture into a procedural description of software components. Information obtained from the class-based models, flow models, and behavioral models serve as the basis for component design.

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

Two important principles are profusely used:

- **Abstraction**
- **Decomposition**

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:

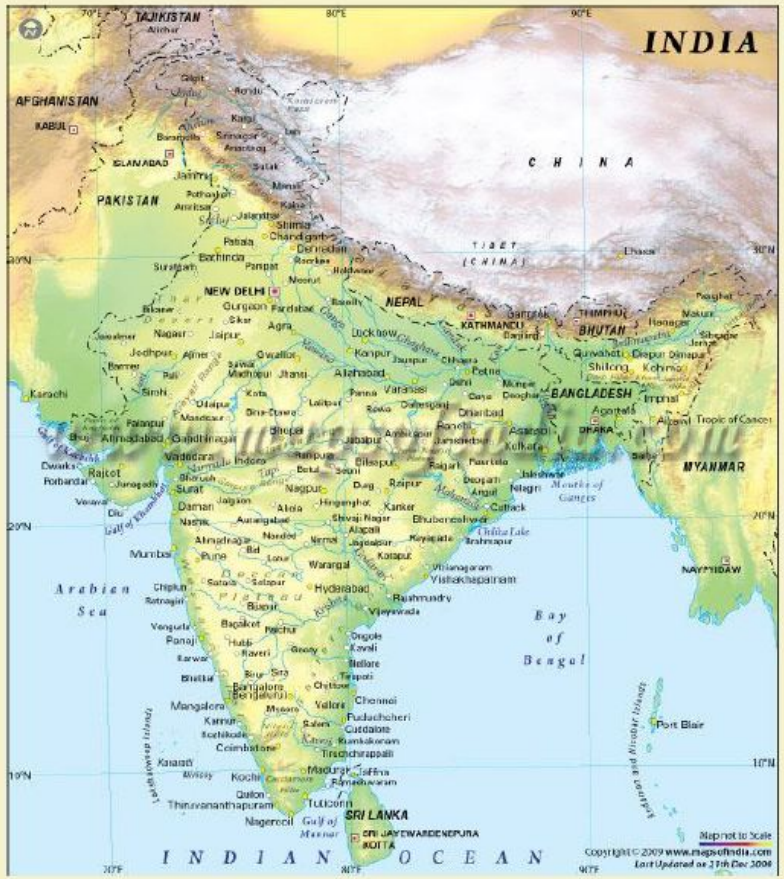
  - Meet all the citizens of the country, visit every house, and examine every tree of the country?

– You would possibly refer to various types of maps for that country only.

You would study an Abstraction…

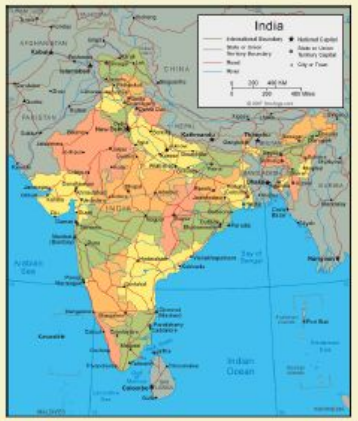
A map is:

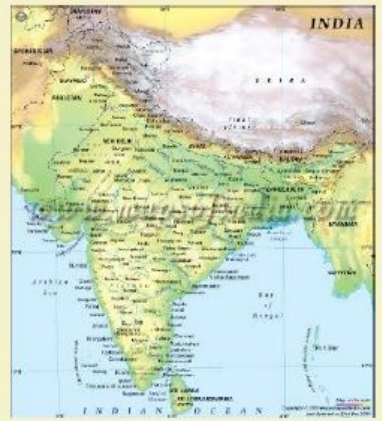
- An abstract representation of a country.
- Various types of maps (abstractions) possible.



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.

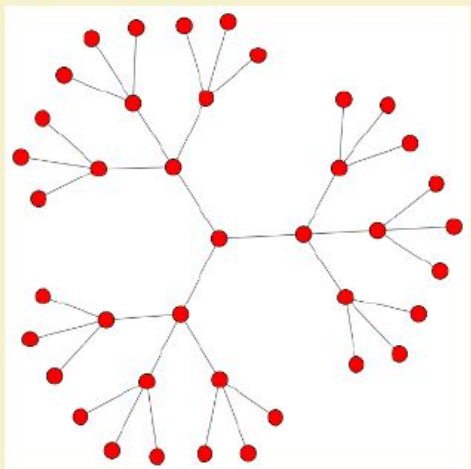
Abstractions of Complex Problems

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.



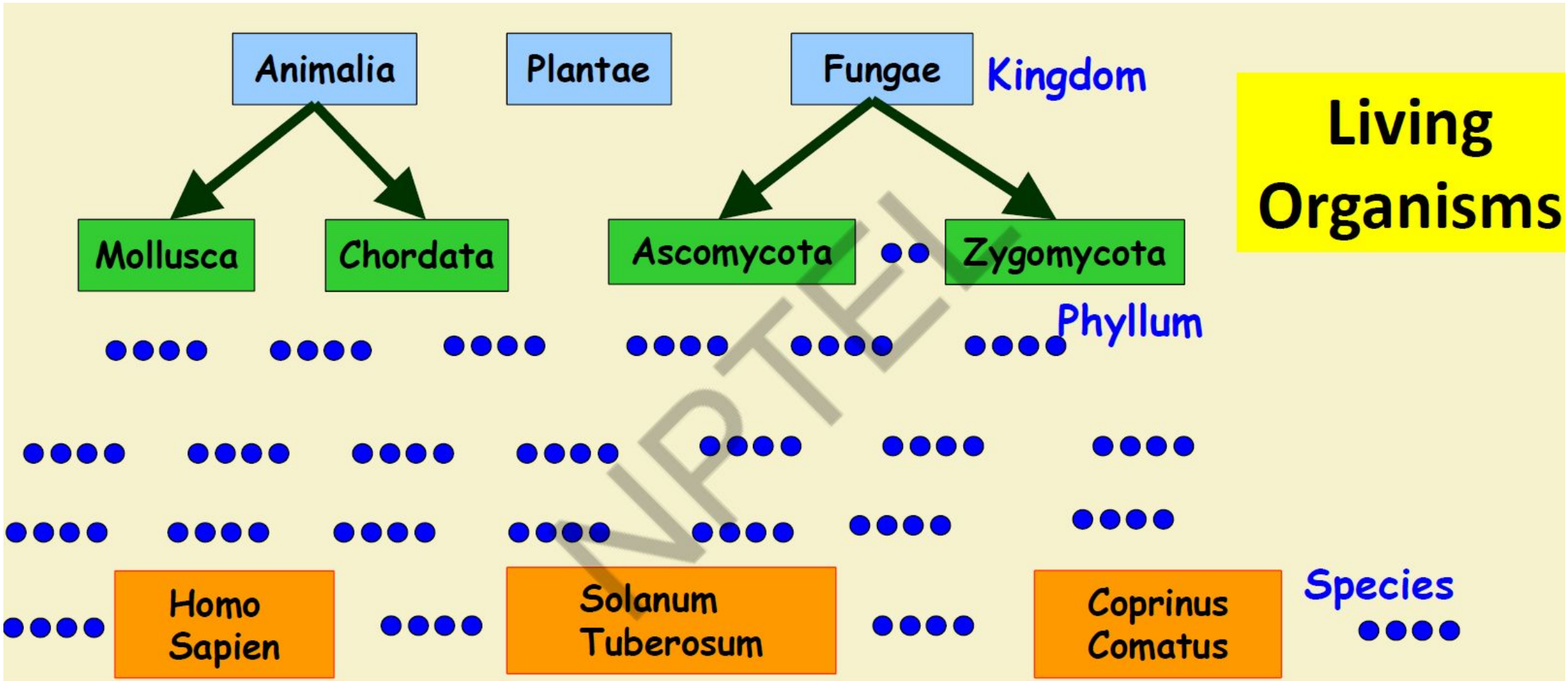
Abstraction of Complex Problems -- An Example

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.


Animalia
Plantae
Fungae
Kingdom
Living Organisms
Mollusca
Chordata
Ascomycota
Zygomycota
Phyllum
Homo Sapien
Solanum Tuberosum
Coprinus Comatus
Species

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

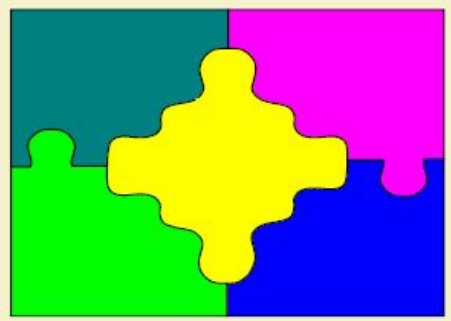
Example use of decomposition principle:

- 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:

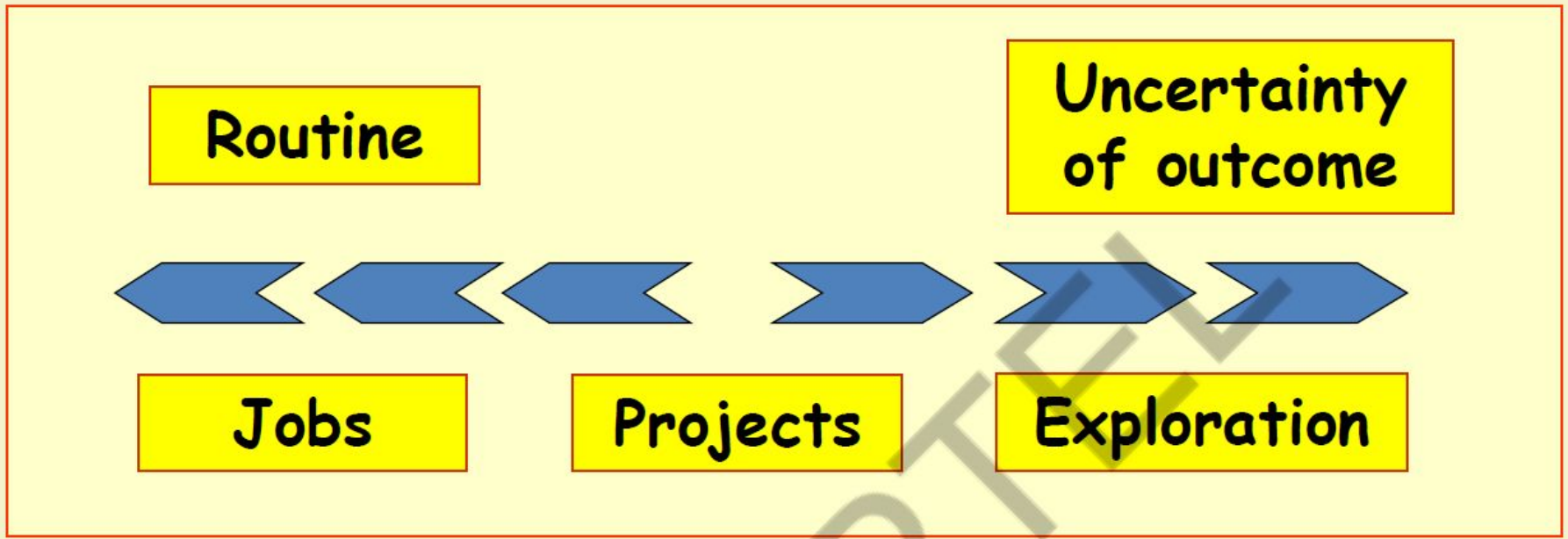
- **Specification, design, user interface development, testing, project management, maintenance, etc.**

- 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:

Types of Software Projects

- **Products (Generic software)**
- **Services (custom software)**

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

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

Scenario of Indian Software Companies

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

Structured Programming

A program is called structured:

–When it uses only the following types of constructs:

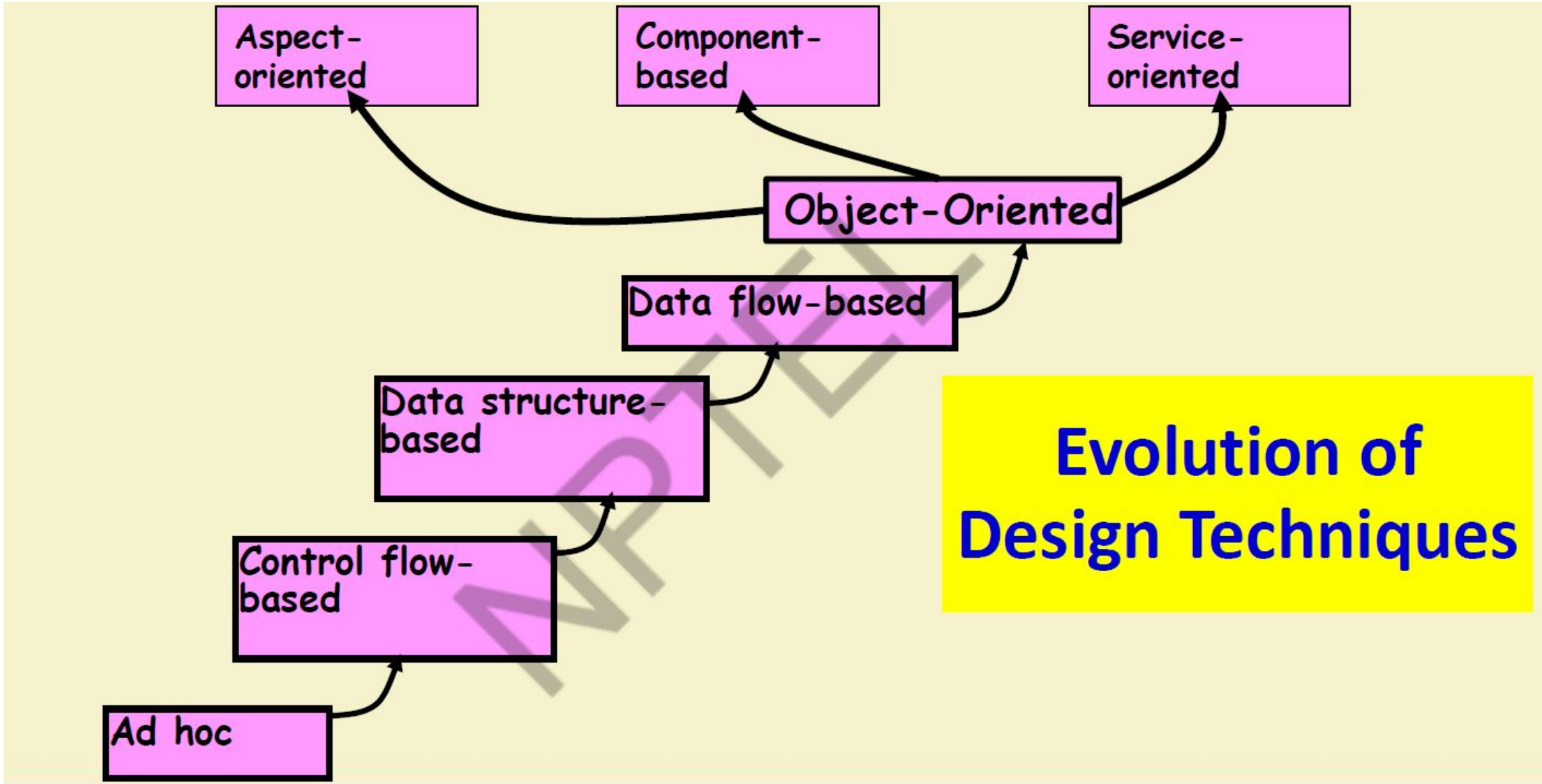
- **sequence,**
- **selection,**
- **iteration**

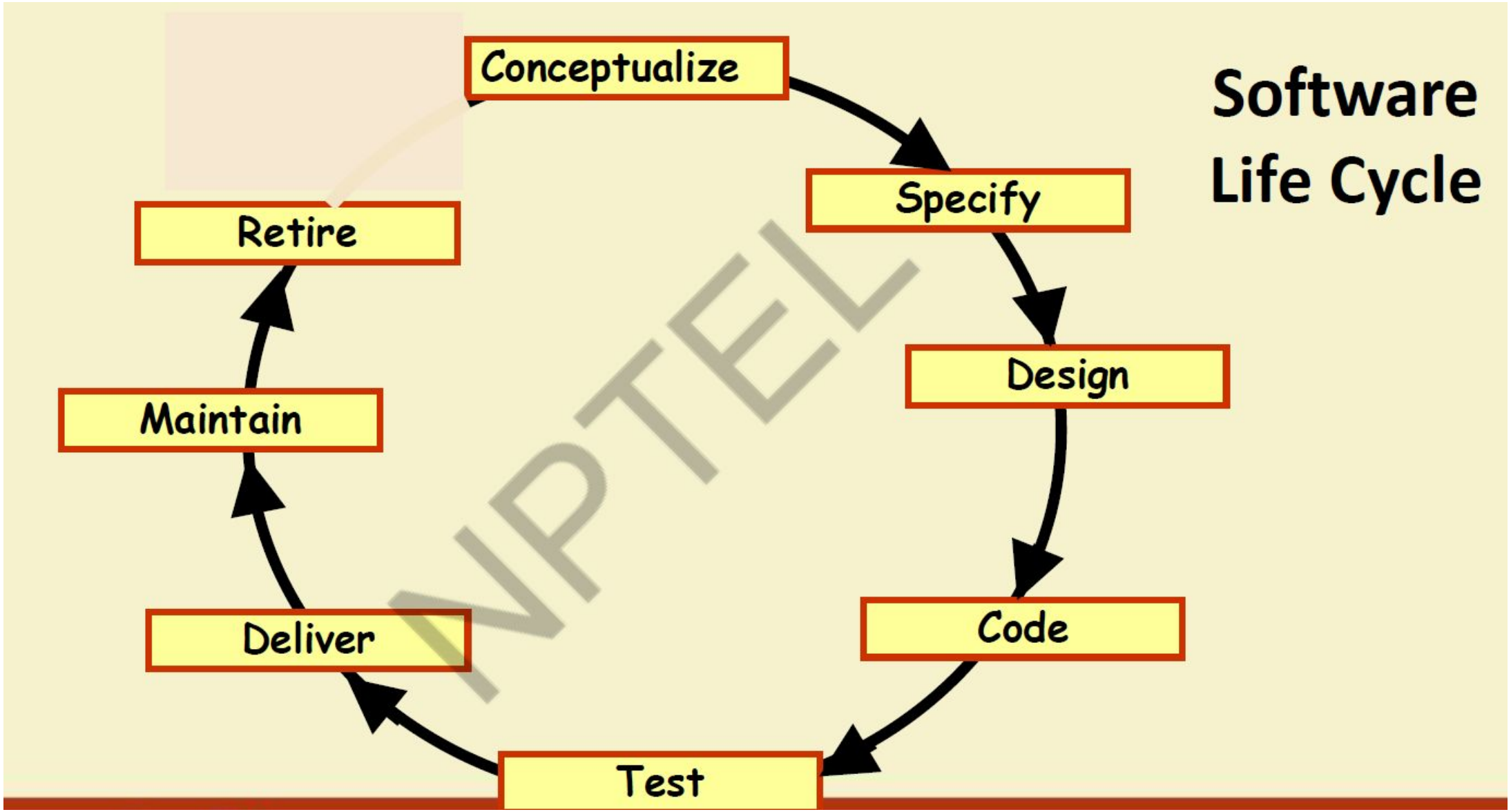
–Consists of modules.

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.


Aspect-oriented
Component-based
Service-oriented
Object-Oriented
Data flow-based
Data structure-based
Control flow-based
Ad hoc
Evolution of Design Techniques


Software
Life Cycle
Conceptualize
Specify
Design
Code
Test
Deliver
Maintain
Retire
NPTEL

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.**

Life Cycle Model (CONT.)

Many life cycle models have been proposed.

We confine our attention to only a few commonly used models.

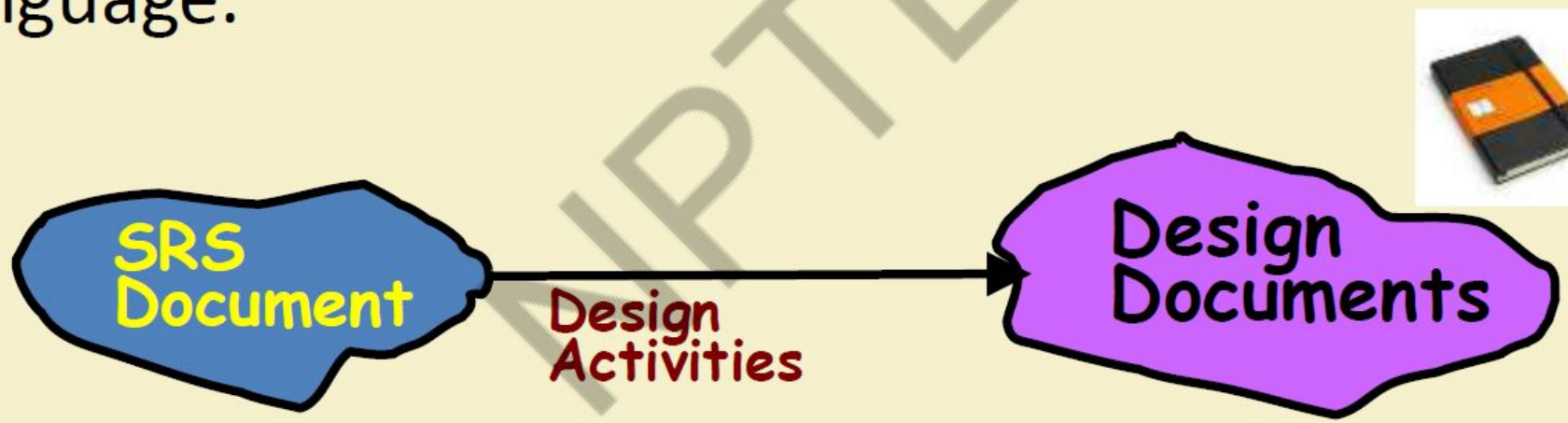
- Waterfall
- V model,
- Evolutionary,
- Prototyping
- Spiral model,
- Agile models

Traditional models

What is Achieved during design phase?

Transformation of SRS document to Design document:

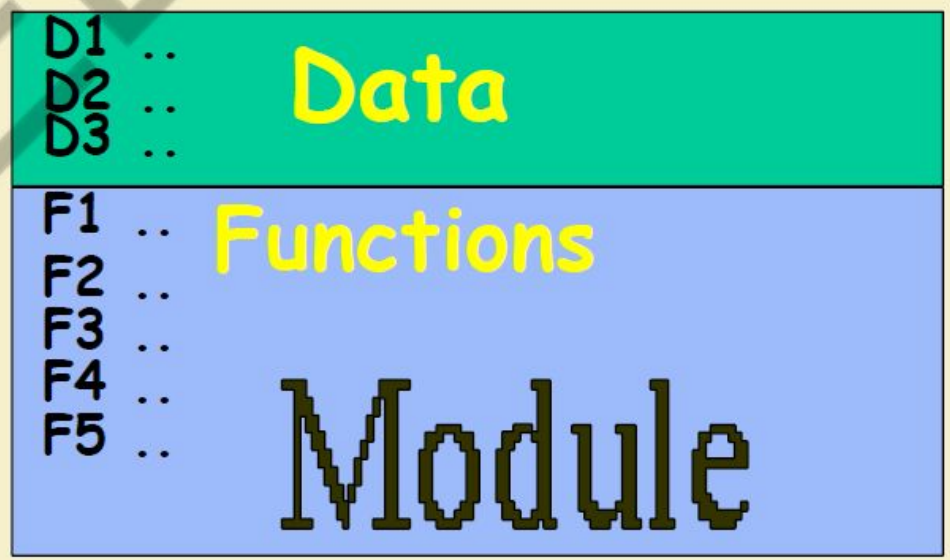
– A form easily implementable in some programming language.



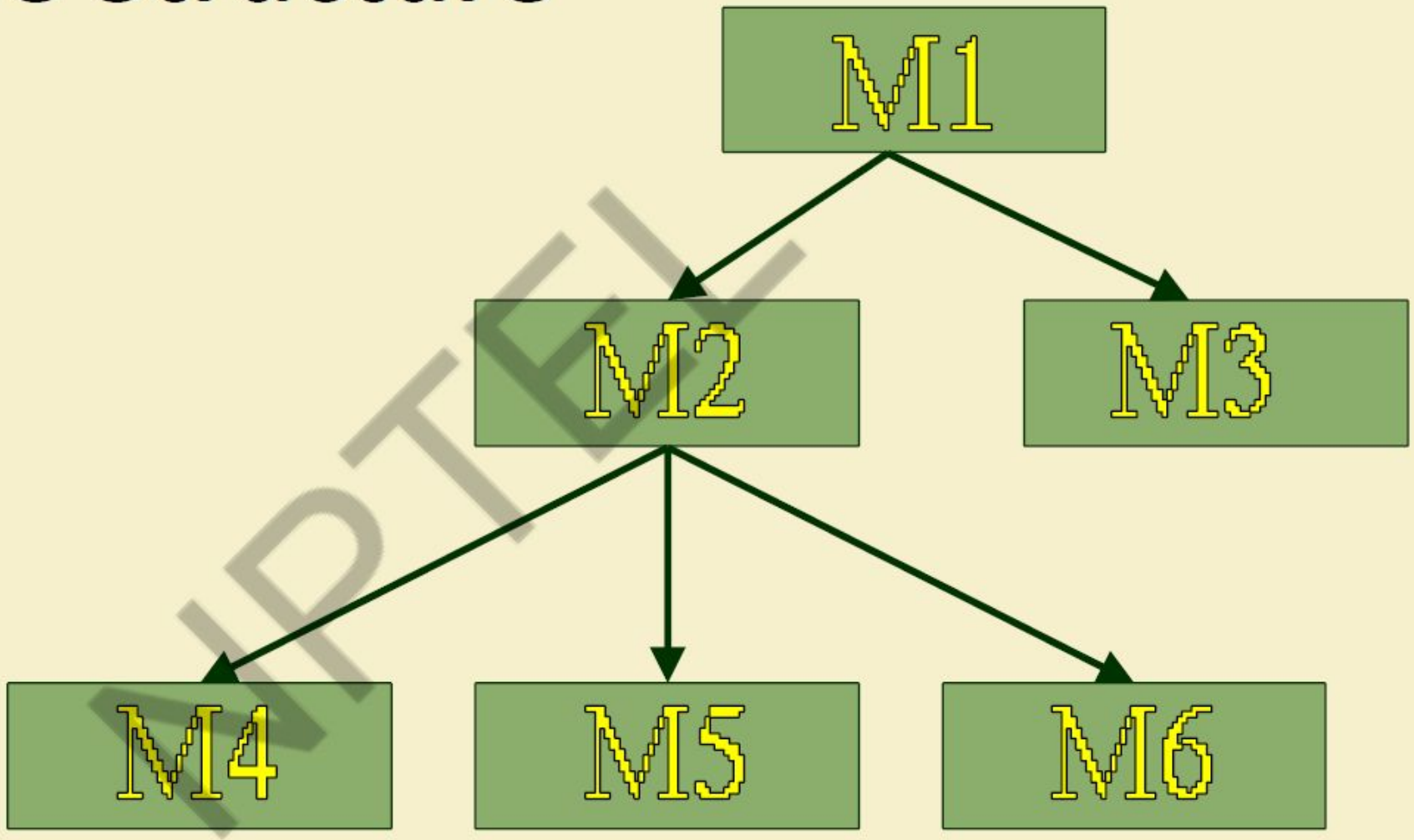
Module

A module consists of:

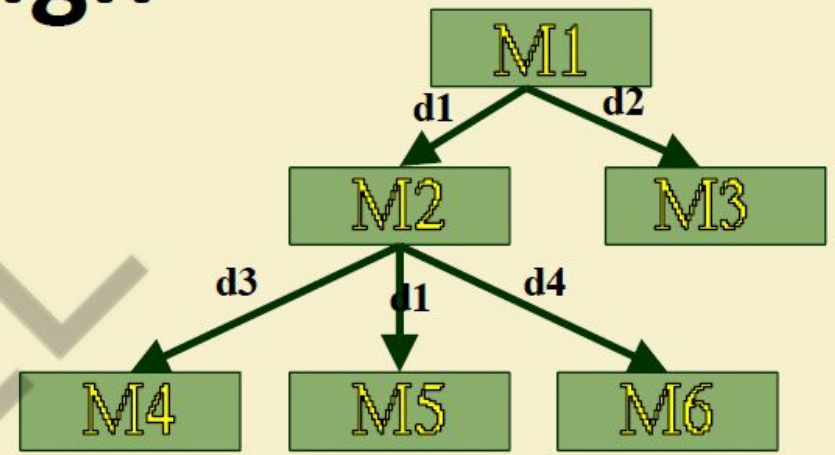
- several functions
- associated data structures.



Module Structure



High-level design



Identify:

- modules
- control relationships among modules
- interfaces among modules.

Detailed design

For each module, design for it:

- data structure
- algorithms

Outcome of detailed design:

- module specification.