SOFTWARE DESIGN

Dr.L.M.Nithya,

Professor & Dean-IT

SOFTWARE DESIGN

- Deriving a solution which satisfies software requirements

PROGRAMMER'S APPROACH TO SOFTWARE ENGINEERING

Skip requirements engineering and design phases; start writing code

WHY THIS PROGRAMMER'S APPROACH?

- Design is a waste of time
- We need to show something to the customer real quick
- We are judged by the amount of LOC/month
- We expect or know that the schedule is too tight

WHAT IS NOT DESIGN

- Design is not programming.
- Design is not modeling. Modeling is part of the architectural design.
- Design is not part of requirements.
- Where requirements finishes and where design starts ?.
- Requirements = What the system is supposed to do.
- Design = How the system is built.

WHAT IS DESIGN (OR ARCHITECTURE?)

- A high-level model of a software system
  - Describes the structure, functionality and characteristics of the software system.
  - Understandable to many stakeholders
  - Allows evaluation of the system's properties before it is built
  - Provides well understood tools and techniques for constructing the thing from its blueprint
- A software system's blueprint
  - Its components
  - Their interactions
  - Their interconnections
- Which aspects of a software system are architecturally relevant?

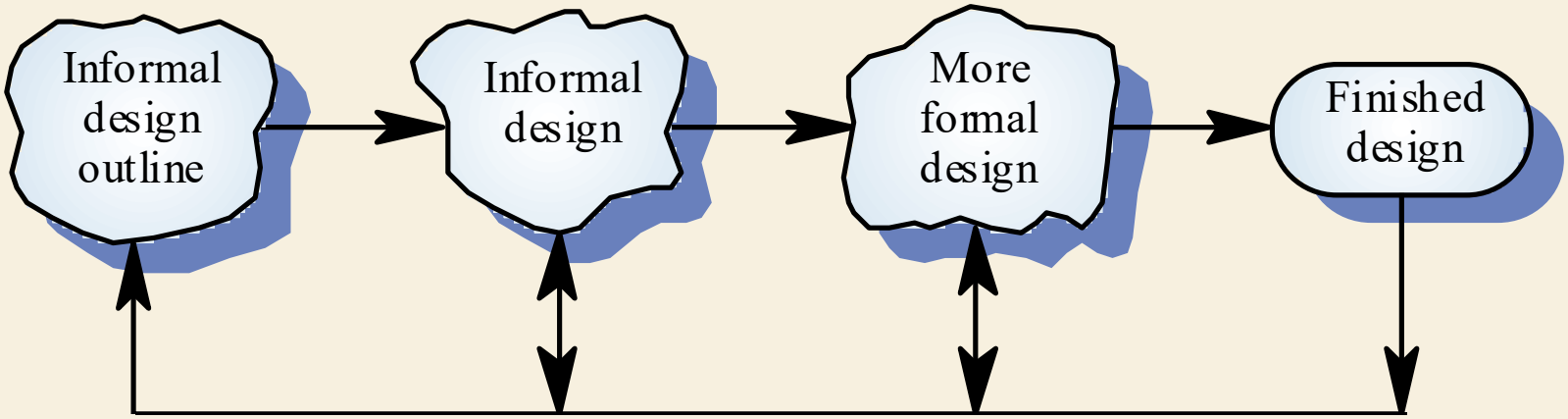
WHAT IS DESIGN (OR ARCHITECTURE?)

- How should they be represented most effectively to enable stakeholders to understand, reason, and communicate about a system before it is built?
- What tools and techniques are useful for implementing an architecture in a manner that preserves its properties?
- We design the software but we must consider the hardware (and the environment).
- Design must reflect requirements, and we must be able to relate each requirements with parts of the design.
- How can we include non-functional requirements into the design?

STAGES OF DESIGN

- Problem understanding
  - Look at the problem from different angles to discover the design requirements
- Identify one or more solutions
  - Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources
- Describe solution abstractions
  - Use graphical, formal or other descriptive notations to describe the components of the design
- Repeat process for each identified abstraction until the design is expressed in primitive terms

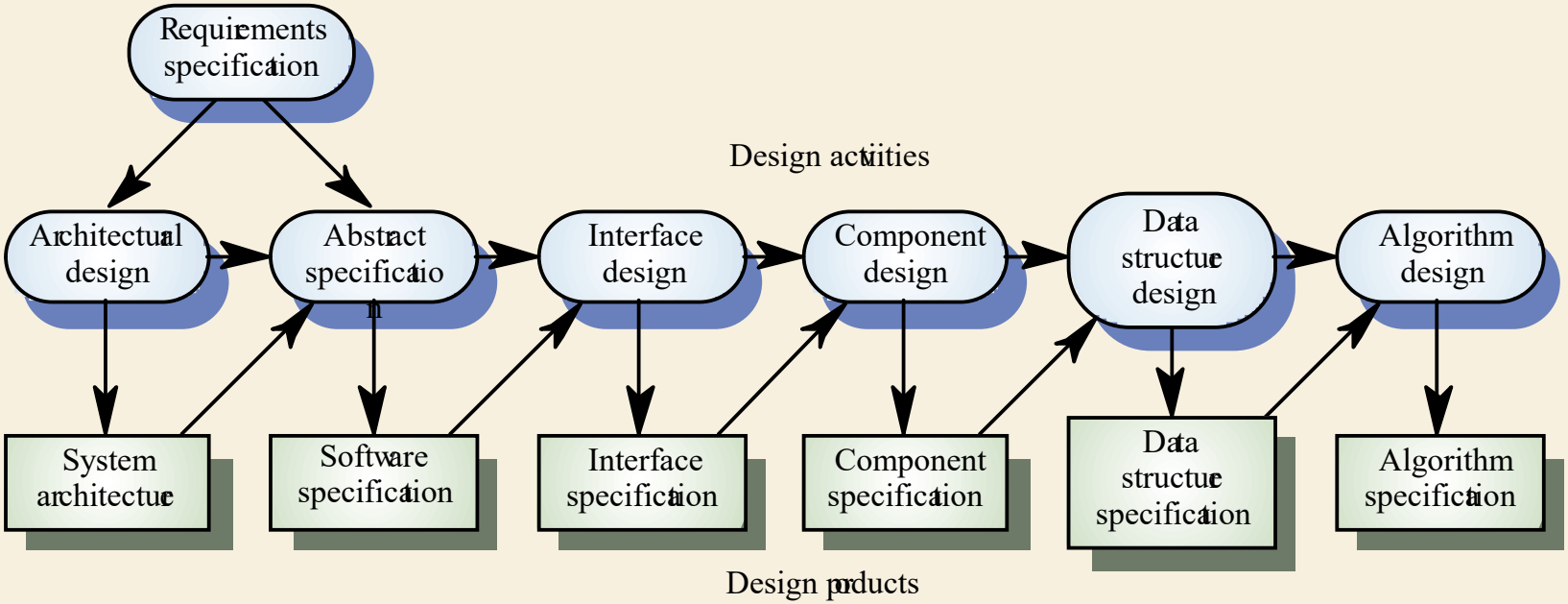
FROM INFORMAL TO FORMAL DESIGN



THE DESIGN PROCESS

- The system should be described at several different levels of abstraction
- Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary

PHASES IN THE DESIGN PROCESS



DESIGN PHASES

- *Architectural design* Identify sub-systems
- *Abstract specification* Specify sub-systems
- *Interface design* Describe sub-system interfaces
- *Component design* Decompose sub-systems into components
- *Data structure design* Design data structures to hold problem data
- *Algorithm design* Design algorithms for problem functions

FROM REQUIREMENTS TO ARCHITECTURE

- From problem definition to requirements specification
  - Determine exactly what the customer and user want
  - Specifies what the software product is to do
- From requirements specification to architecture
  - How do we plan to build (design) the system ?
  - Decompose software into modules with interfaces
  - Specify high-level behavior, interactions, and non-functional properties
  - Consider key tradeoffs
    - Schedule vs. Budget
    - Cost vs. Robustness
    - Fault Tolerance vs. Size
    - Security vs. Speed
  - Maintain a record of design decisions and traceability
  - Specifies how the software product is to do its tasks (from design to programming).

ARCHITECTURAL DESIGN

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Where do we finish requirements and start design ?.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

ADVANTAGES OF EXPLICIT ARCHITECTURE

- From Requirements to Design
- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.
- From Design to Programming ,Testing & Maintenance.

ARCHITECTURE AND SYSTEM CHARACTERISTICS

How the system must be designed to achieve:

- Performance
- Security
- Safety
- Reliability
- Availability
- Maintainability
- Quality

ARCHITECTURAL DESIGN PROCESS

- System structuring
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- Control modelling
  - A model of the control relationships between the different parts of the system is established
- Modular decomposition
  - The identified sub-systems are decomposed into modules

DESIGN QUALITY

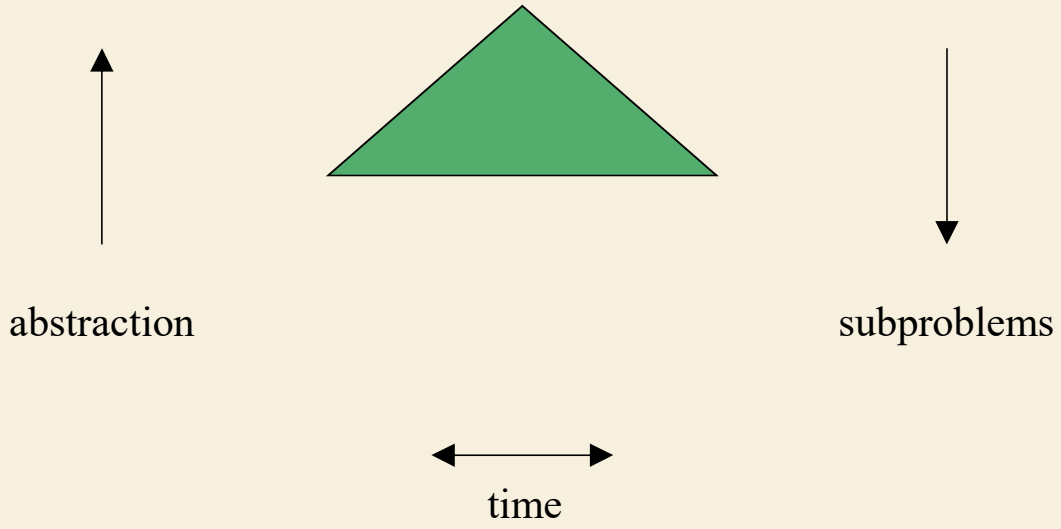
- Design quality is an elusive concept. Quality depends on specific organisational priorities
- A 'good' design may be the most efficient, the cheapest, the most maintainable, the most reliable, etc.
- The attributes discussed here are concerned with the maintainability of the design
- Quality characteristics are equally applicable to function-oriented and object-oriented designs

DESIGN PRINCIPLES

- Abstraction
- Modularity, coupling and cohesion
- Information hiding
- Limit complexity
- Hierarchical structure
- Understandability
- Adaptability

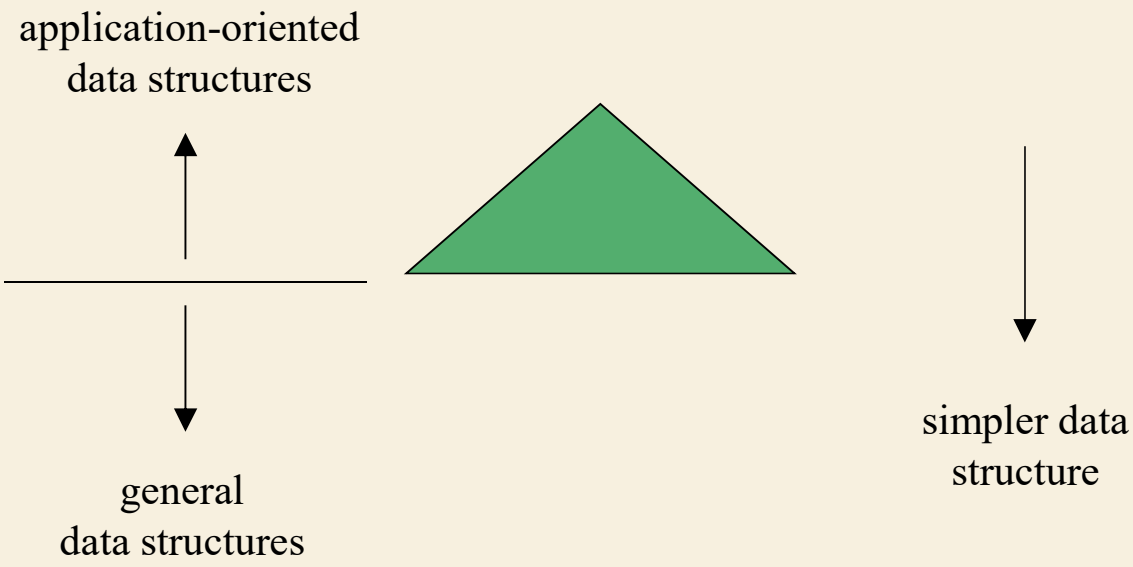
ABSTRACTION

- procedural abstraction: natural consequence of stepwise refinement: name of procedure denotes sequence of actions



ABSTRACTION

- data abstraction: aimed at finding a hierarchy in the data



MODULARITY

- structural criteria which tell us something about individual modules and their interconnections
- cohesion and coupling
- cohesion: the glue that keeps a module together
- coupling: the strength of the connection between modules

COHESION

- A measure of how well a component 'fits together'
- A component should implement a single logical entity or function
- Cohesion is a desirable design component attribute as when a change has to be made, it is localised in a single cohesive component
- Various levels of cohesion have been identified

COHESION LEVELS

- Coincidental cohesion (weak)
  - Parts of a component are simply bundled together
- Logical association (weak)
  - Components which perform similar functions are grouped
- Temporal cohesion (weak)
  - Components which are activated at the same time are grouped
- Procedural cohesion (weak)
  - The elements in a component make up a single control sequence

COHESION LEVELS

- Communicational cohesion (medium)
  - All the elements of a component operate on the same input or produce the same output
- Sequential cohesion (medium)
  - The output for one part of a component is the input to another part
- Functional cohesion (strong)
  - Each part of a component is necessary for the execution of a single function
- Object cohesion (strong)
  - Each operation provides functionality which allows object attributes to be modified or inspected

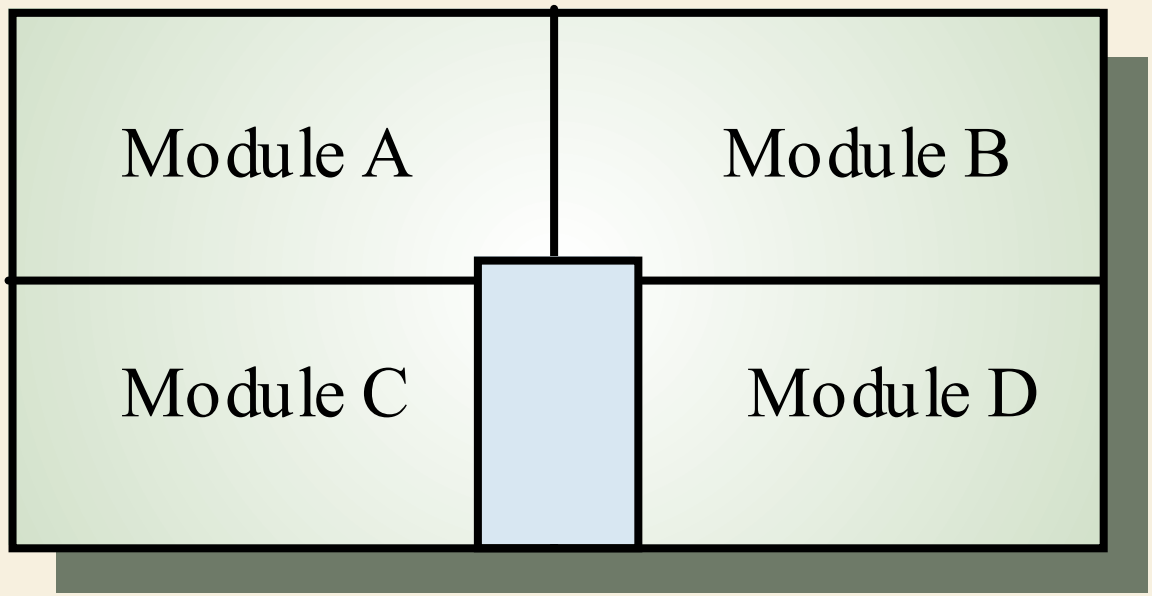
COHESION AS A DESIGN ATTRIBUTE

- Not well-defined. Often difficult to classify cohesion
- Inheriting attributes from super-classes weakens cohesion
- To understand a component, the super-classes as well as the component class must be examined
- Object class browsers assist with this process

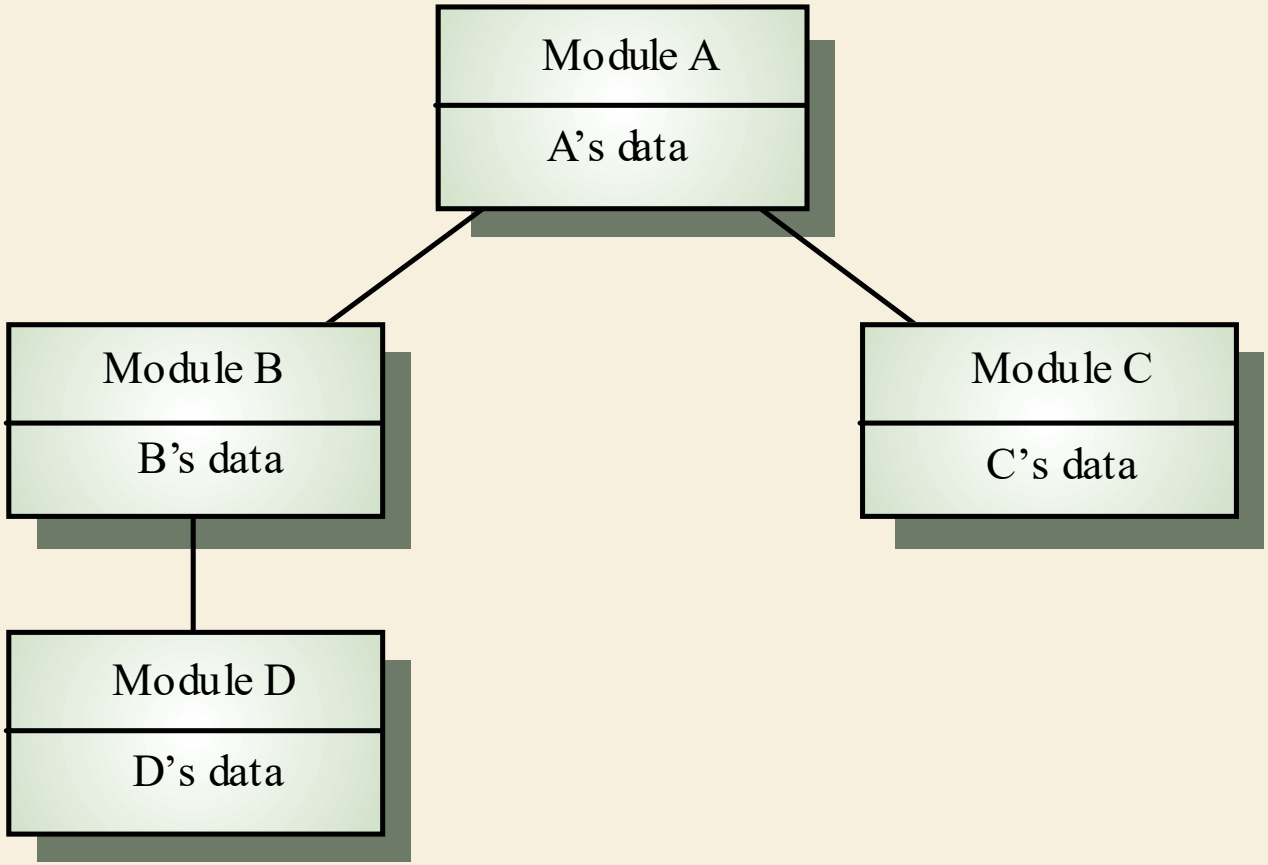
COUPLING

- A measure of the strength of the inter-connections between system components
- Loose coupling means component changes are unlikely to affect other components
- Shared variables or control information exchange lead to tight coupling
- Loose coupling can be achieved by state decentralisation (as in objects) and component communication via parameters or message passing

TIGHT COUPLING



LOOSE COUPLING



COUPLING AND INHERITANCE

- Object-oriented systems are loosely coupled because there is no shared state and objects communicate using message passing
- However, an object class is coupled to its super-classes. Changes made to the attributes or operations in a super-class propagate to all sub-classes. Such changes must be carefully controlled

INFORMATION HIDING

- each module has a secret
- design involves a series of decision: for each such decision, wonder who needs to know and who can be kept in the dark
- information hiding is strongly related to
  - abstraction: if you hide something, the user may abstract from that fact
  - coupling: the secret decreases coupling between a module and its environment
  - cohesion: the secret is what binds the parts of the module together

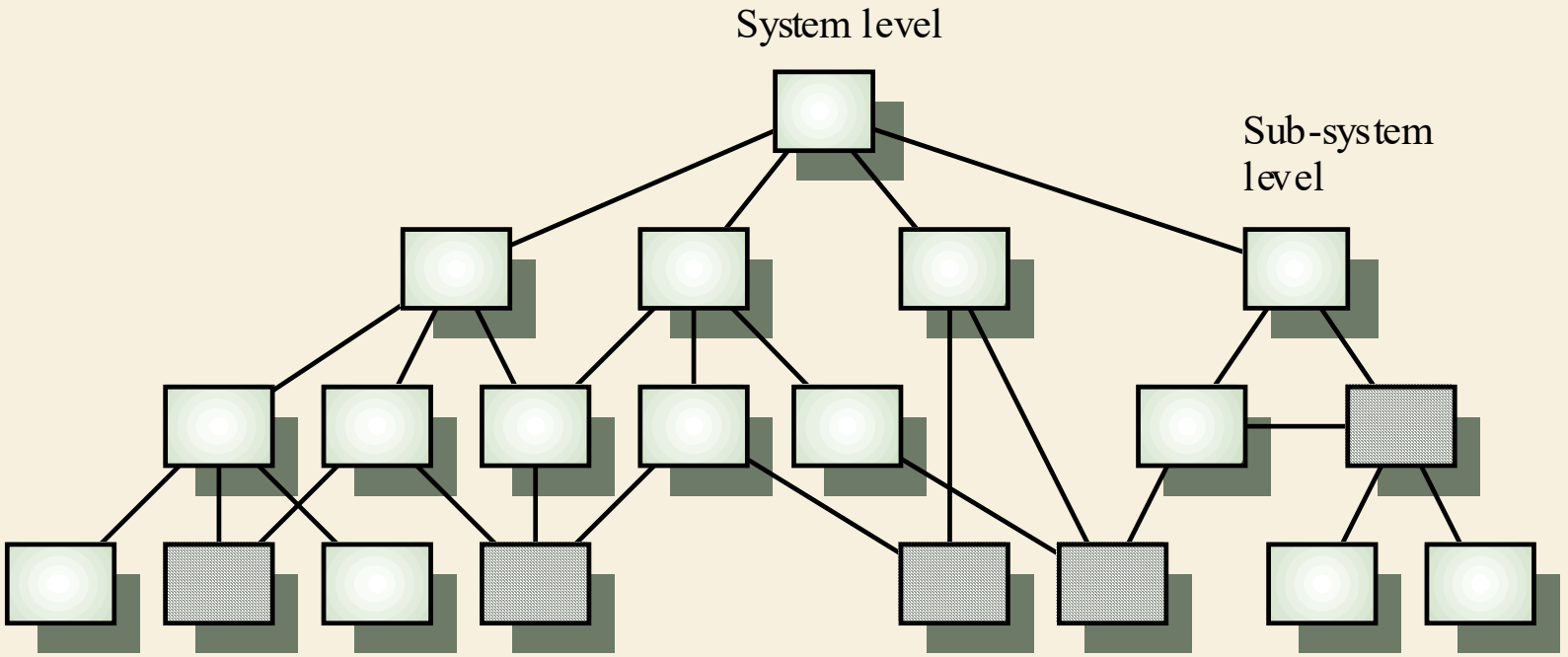
COMPLEXITY

- measure certain aspects of the software (lines of code, # of if-statements, depth of nesting, …)
- use these numbers as a criterion to assess a design, or to guide the design
- interpretation: higher value ⇒ higher complexity ⇒ more effort required (= worse design)
- two kinds:
  - **intra-modular:** inside one module
  - **inter-modular:** between modules

TOP-DOWN DESIGN

- In principle, top-down design involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level
- In practice, large systems design is never truly top-down. Some branches are designed before others. Designers reuse experience (and sometimes components) during the design process

HIERARCHICAL DESIGN STRUCTURE



UNDERSTANDABILITY

- Related to several component characteristics
  - *Cohesion*. Can the component be understood on its own?
  - *Naming*. Are meaningful names used?
  - *Documentation*. Is the design well-documented?
  - *Complexity*. Are complex algorithms used?
- Informally, high complexity means many relationships between different parts of the design. hence it is hard to understand
- Most design quality metrics are oriented towards complexity measurement. They are of limited use

ADAPTABILITY

- A design is adaptable if:
  - Its components are loosely coupled
  - It is well-documented and the documentation is up to date
  - There is an obvious correspondence between design levels (design visibility)
  - Each component is a self-contained entity (tightly cohesive)
- To adapt a design, it must be possible to trace the links between design components so that change consequences can be analysed

DESIGN TRACEABILITY