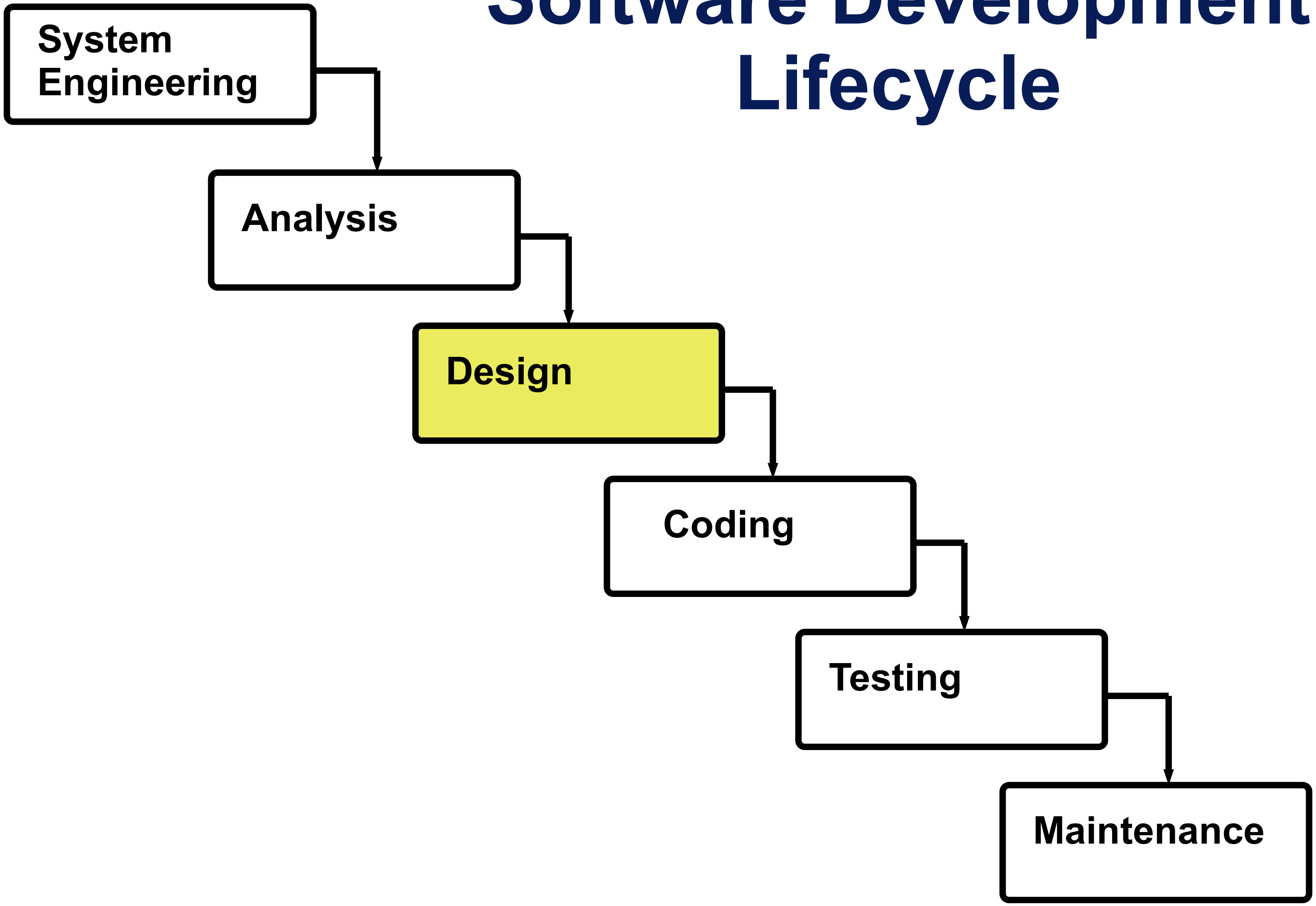
SOFTWARE DESIGN

- Software Design Fundamentals
- Data Flow-Oriented Design
- Object-Oriented Design
- Data-Oriented Design Methods
- User Interface Design
- Real-Time Design

Software Development Lifecycle



Software Design ...

- is the first step in the development phase for any engineered system
- produces a model of the software which is to be coded later

"The beginning of wisdom for a computer programmer is to recognize the difference between getting a program to work and getting it right."

-- M.A. Jackson, *Principles of Program Design*, 1975

Design Models

- Architectural Design - Relationship among major structural components of the program.
- Data Design - Transforms the information domain model created during analysis into the data structures required to implement the software.
- Procedural Design - Transforms structural components into a procedural description of the software.

Software design requires all three design models

Software Design Steps

1. **Preliminary Design - The transformation of requirements into a data and procedural architecture.**
2. **Detailed Design - Refining the architectures developed in preliminary design.**

The idea is to transform the structure and details from the problem domain to the implementation domain sufficient for coding.

Quality

Design is the phase where *quality* is built into software.

The quality of an evolving design is identified through a series of formal technical reviews.

Guidelines for a Good Design

- A design should exhibit a hierarchical organization.
- A design should be modular, leading to an implementation of strongly cohesive, loosely coupled modules.
- A design should contain a distinct and separate representation of data and procedure.
- A design should be derived using a repeatable method driven by information obtained from the requirements analysis.
- A design should track closely with the requirements - there should be a mapping.

Fundamental Concepts

- ***Stepwise Refinement*** **- the successive definition of levels of detail**
- ***Software Architecture*** **- the hierarchical structure of procedural components and the structure of data**
- ***Program Structure*** **- the flow of control between the procedural components**
- ***Software Procedure*** **- the processing details of each procedural component**
- ***Data Structure*** **- the logical relationship between elements of data**
- ***Levels of Abstraction*** **- the expression of a design in terms of the problem space, usually employing *Stepwise Refinement* in the process**
- ***Information Hiding*** **- the suppression of unnecessary details at a particular level of abstraction**

Diagramming Techniques

Many of the diagramming techniques used during requirements analysis may also be used during design:

- **Data Flow diagrams**
- **State Transition diagrams**
- **Entity-Relationship diagrams**

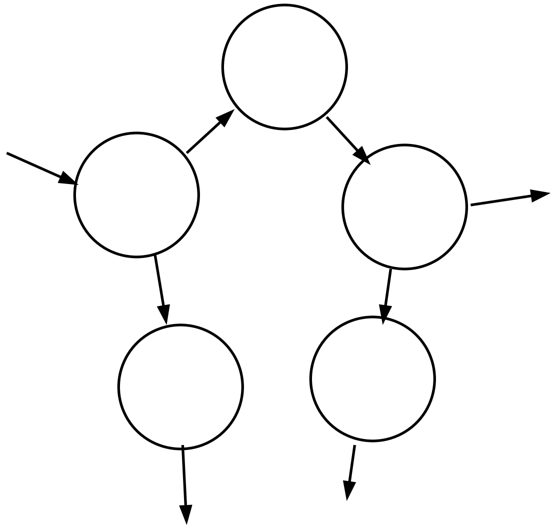
We add several more types of diagrams to specifically support software structure:

- **Structure Charts**
- **Function diagrams (also called flow-diagrams)**

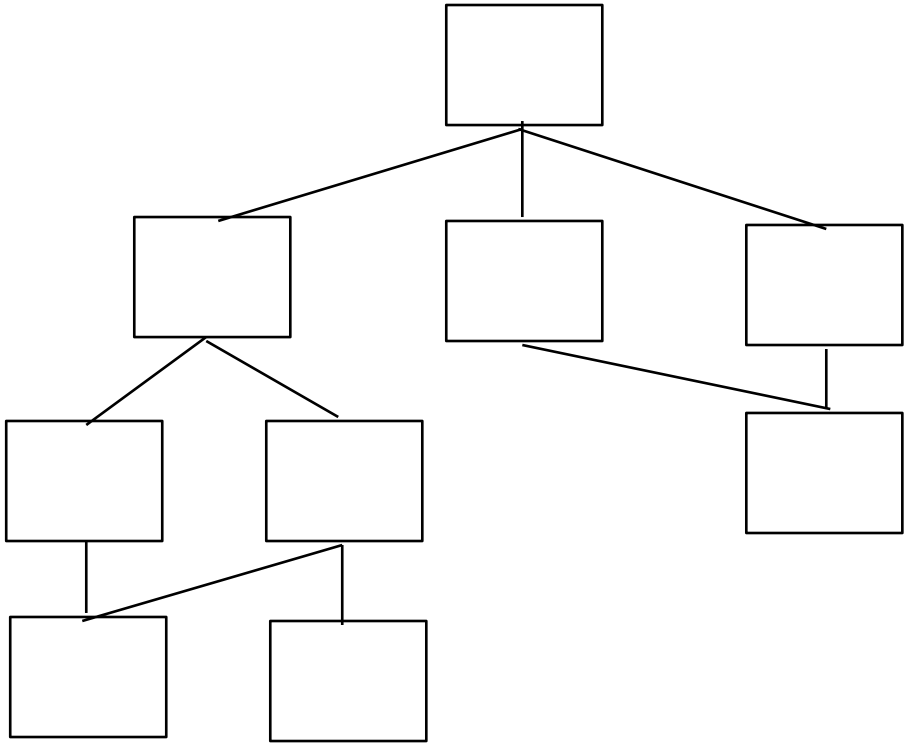
Other diagramming techniques are intended specifically for design and are often language-specific. These techniques are often used when the implementation language supports object oriented programming such as Ada or C++:

- **Object Interaction diagrams**
- **Booch diagrams**

Evolution of Structure

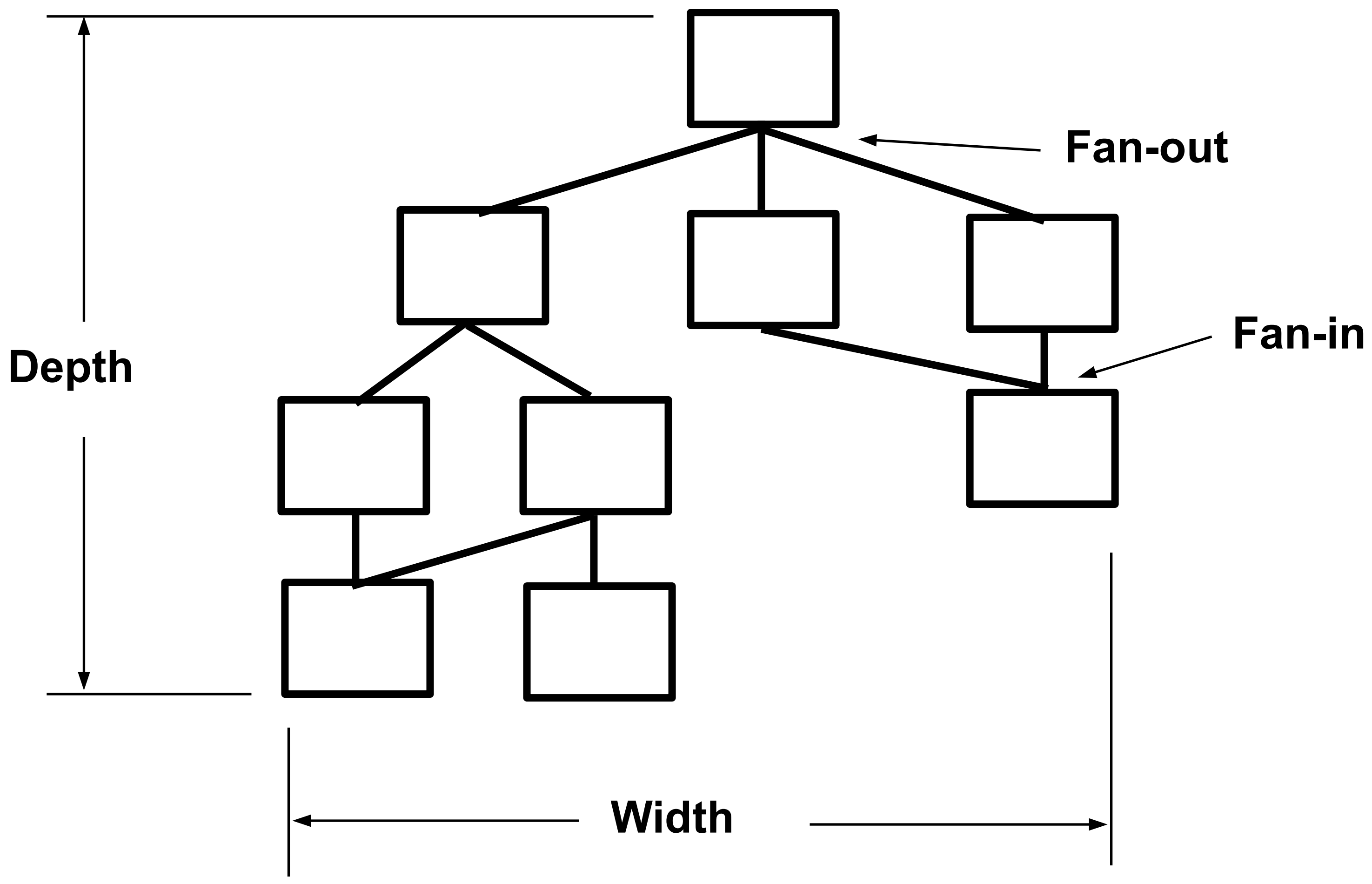


DFD

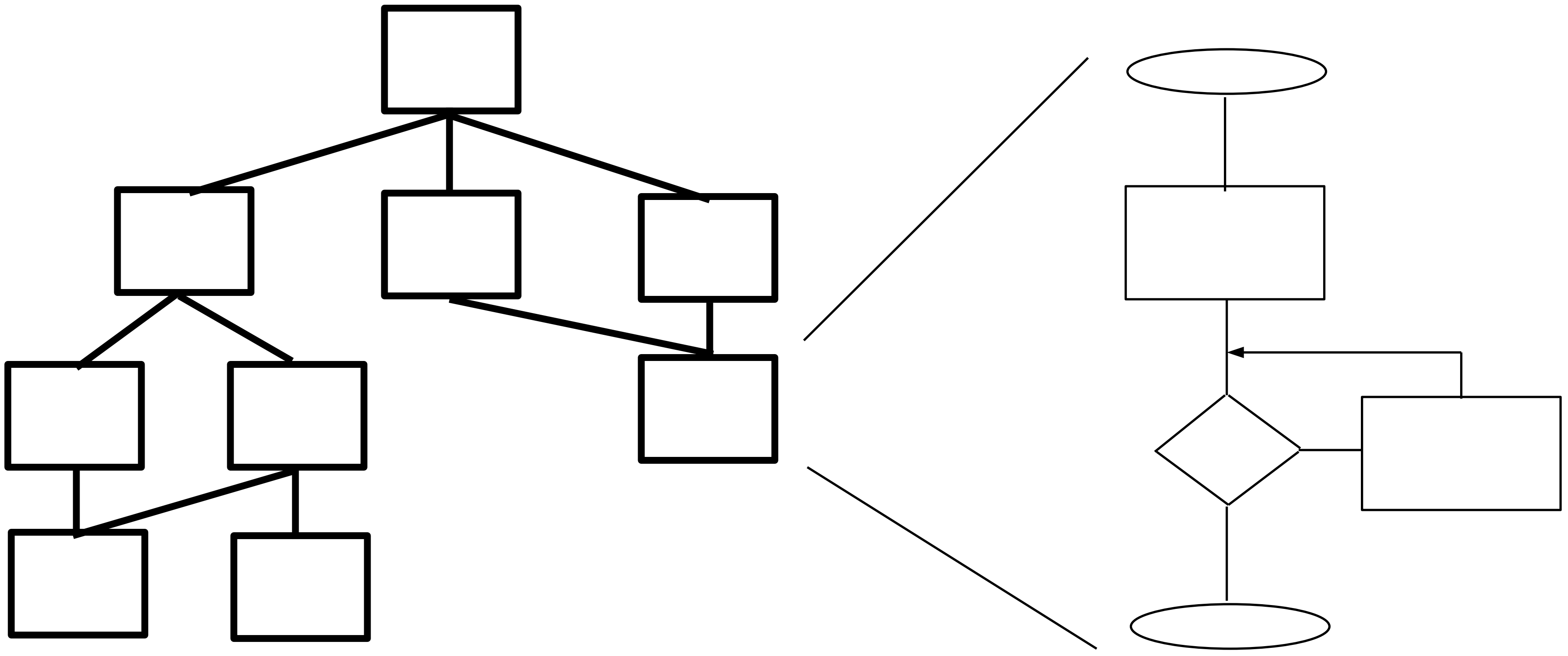


Structure Chart

Structure Chart Notation



Modules



Structure Chart

Flow Diagram

Modular Design

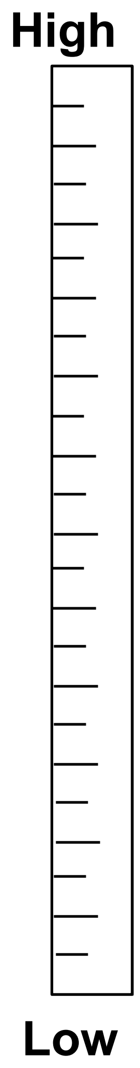
There are three basic types of modules:

- *Sequential* - referenced and executed without apparent interruption
- *Incremental* - can be interrupted by other software prior to completion and restarted at the point of interruption
- *Parallel* - executes concurrently with other modules

As an example, Ada provides features (sometimes independent of the operating system) which directly support the design and coding of these types of modules:

- procedures and functions
- tasks with entry points tied to interrupts
- tasks which may be executed concurrently

Cohesion Spectrum



Functional - module performs one distinct procedural task.

Sequential - module performs sequence of procedural tasks.

Communicational - module performs all tasks on a single area of a data structure.

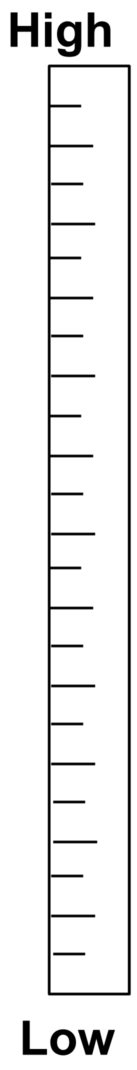
Procedural - procedural tasks are related and performed in some order.

Temporal - All procedural tasks must be performed within a given span of time.

Logical - All procedures have some logical relationship.

Coincidental - No relationship exists between the tasks in the module.

Coupling Spectrum



Content - modules make use of data or control info from each other or has branches into middle of module.

Common - modules commonly reference a global data area.

External - modules regularly reference an external environment like I/O or comm protocol.

Control - modules regularly pass control info between each other, but data access outside of modules is infrequent.

Stamp - All, or part, of data structures passed between modules rather than single-value arguments.

Data - Simple, single-vallues arguments passed between modules.

No direct coupling - modules do not communicate with each other.

Desirable Attributes of Modules

- *Functional Independence* - the isolation of particular functions to particular modules
- *Cohesion* - the binding of a single task to a single module without interaction with or side effects from other modules; *Strong Cohesion* is desirable
- *Coupling* - a measure of the interconnection between modules; *Loose Coupling*, usually implemented by exclusive use of interfaces through subprograms, is desirable

Design Documentation

The documentation of a design should include the following information:

- A description of the design
  - A description of the data, including the data flow and data structure
  - A description of the program structure
  - A description of interfaces within the program structure
  - A description of interfaces between the program and other elements in its environment
- A description of each module
- A description of the structure and details of the global data and files
- Test provisions
- A cross-reference between the design and the requirements which drove the design

DI-MCCR-80012A

DoD-STD-2167A Software Design Document

- Preliminary Design
    - CSCI Overview, including architecture, system states, and memory and processing time constraints
    - CSCI Design Description, including descriptions of the component CSCs
- Detailed Design
    - CSC Design and Constraints, including I/O data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of external elements, logical flow, data structures, local data files or database
    - Global CSCI data and data files
- Requirements Traceability

Evaluation Criteria for Designs

- Internal consistency
- Understandability
- Traceability to requirements documents
- Appropriate analysis, design, or coding techniques used
- Appropriate allocation of sizing and timing resources
- Adequacy of requirements allocation for the CSCIs and CSCs
- Consistency between data definition and data use
- Accuracy and required precision of constants and variables

CASE Tools often support the developing of designs by providing automated checking of these and other criteria.

Design Methodologies
Data Flow-Oriented Design

- Data Flow-Oriented Design
- Data Structure-Oriented Design
- Object-Oriented Design
- Real-Time Design

Note

The first three classes are heavily driven by the *Information Domain.*

Data Flow-Oriented Design

- Uses information flow characteristics to derive the program structure
- There are two design analysis techniques:
  - *Transform Analysis and Design* - the information flow exhibits distinct boundaries between incoming and outgoing data (i.e., input, processing, and output are the three key elements of the data flow)
  - *Transaction Analysis and Design* - an information item causes the flow to branch along a choice of paths
- Data Flow Diagrams (DFD's) are the common graphical means to represent the flow of data

Transform Analysis and Design

Design Steps:

- **Review the fundamental system model**
- **Review and refine the DFD's for the software**
- **Determine the transform and transaction characteristics of the DFD's**
- **Isolate the transform center by specifying incoming and outgoing flows**
- **Perform "first-level factoring" - derive the mapping from the major parts of the DFD to a program structure**
- **Perform "second-level factoring" - map individual bubbles in the DFD into modules in the program structure**
- **Refine the above "first-cut" program structure - maximize cohesion, minimize coupling, and build a structure hierarchy**

Transaction Analysis and Design

Design Steps:

- **Review the fundamental system model**
- **Review and refine the DFD's for the software**
- **Determine the transform and transaction characteristics of the DFD's**
- **Isolate the transaction center and the flow characteristics of each action path**
- **Map the DFD into a software structure amenable to transaction processing**
- **Factor and refine the transaction structure and the structure of each action path**
- **Refine the above "first-cut" program structure - maximize cohesion, minimize coupling, and build a structure hierarchy**

Design Heuristics

- Minimize coupling and maximize cohesion
- Minimize fan-out and strive for fan-in as the depth increases
- Minimize side-effects; keep the scope of the effect of a module within the scope of control of that module
- Evaluate module interfaces to reduce complexity and redundancy; improve consistency of the module
- Define modules whose function is predictable and testable
- Strive for single-entry, single-exit modules
- Package softwawre based on design constraints and portability requirements

Design Methodologies
Data Structure-Oriented Design

- Data Flow-Oriented Design
- Data Structure-Oriented Design
- Object-Oriented Design
- Real-Time Design

Note

The first three classes are heavily driven by the *Information Domain.*

Data Structure-Oriented Design

- Three key methods:
  - *Jackson System Development* - concentrates on process modeling and control
  - *Logical Construction of Programs (Warnier)* - rigorous view of data structure and focus on detailed procedural design
  - *Data Structured System Development (Orr)* - incorporates data flow analysis with the Logical Construction of Programs and Jackson System Development (JSD to a lesser extent)
- This is 1970's technology and is not covered in detail

Design Methodologies
Object-Oriented Design

- **Data Flow-Oriented Design**
- **Data Structure-Oriented Design**
- **Object-Oriented Design**
- **Real-Time Design**

Note

The first three classes are heavily driven by the *Information Domain.*

Object-Oriented Design (OOD)

- Concerns itself with creating a model of the real world
- Objects represent the information domain, and the operations associated with that information are grouped with the objects
- Messages (interfaces) provide a means by which operations are invoked
- Packaging of objects with their associated operations takes place - data and procedural abstractions are combined in a single program component called an *object* or a *package*
- OOD representations are more prone than others to programming language dependency

Terminology Overview

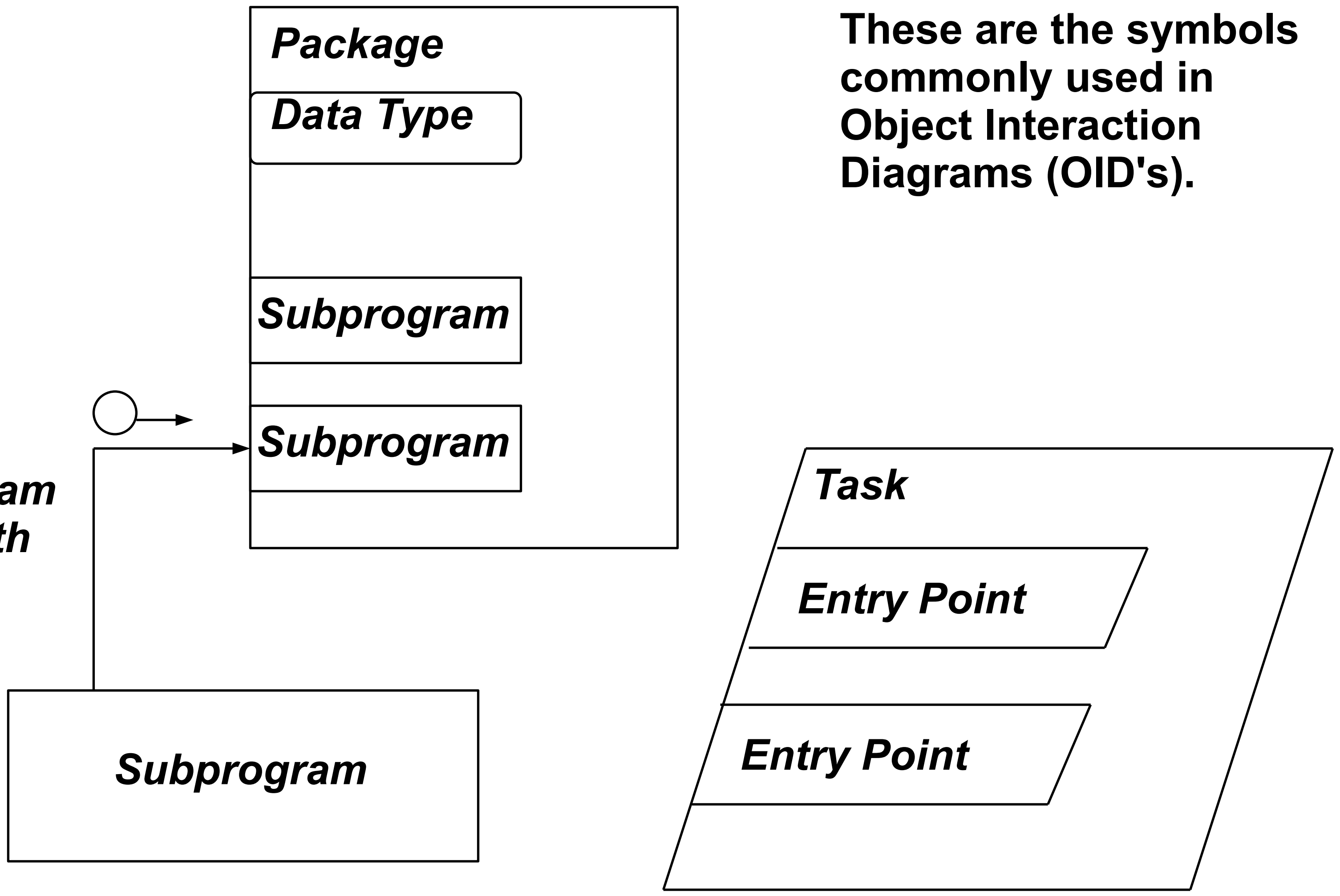
- ***Object*** **- a component of the real world that is mapped into the software domain or an information item**
- ***Operations*** **or** ***Methods*** **- processes which act on objects to transform their internal data structure or provide information on their internal data structures**
- ***Message*** **- a request to an object to perform one of its operations**
- ***Class*** **- a set of objects which share common characteristics**
- ***Instance*** **- an individual object of a class**

Object-Oriented Design Steps

- Identify the objects
- Identfy the attributes of the objects
- Identify the operations that may be applied to the objects
- Establish the interfaces of the objects to the outside world (Ada package specifications may be used if Ada is the implementation language)
- Implement the objects (Ada package bodies may be used if Ada is the implementation language)
- Graphical representation may be employed; Booch Diagrams and Object Interaction Diagrams are the recommended diagramming techniques

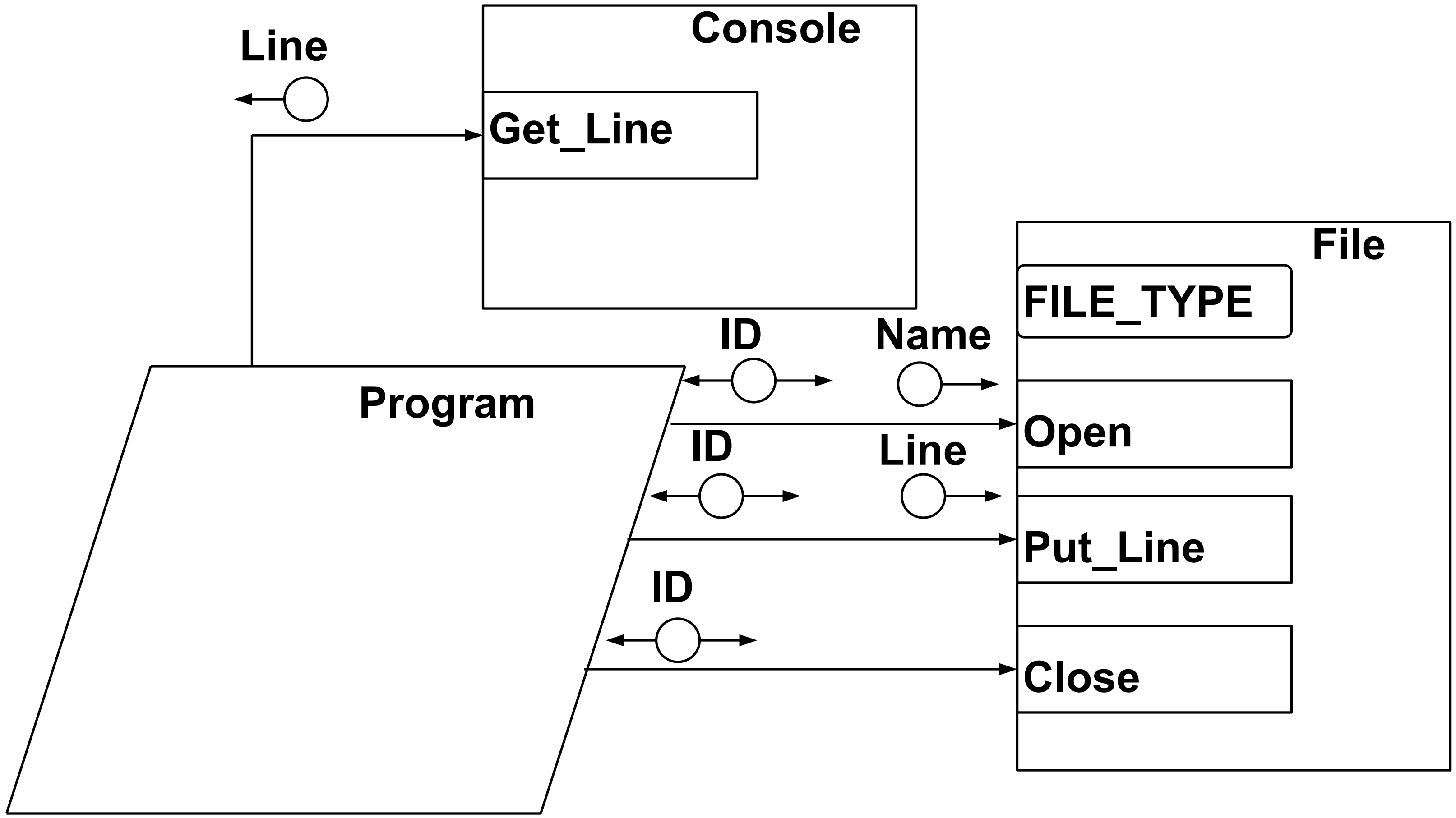
Object Interaction Diagrams (OIDs)

These are the symbols commonly used in Object Interaction Diagrams (OID's).

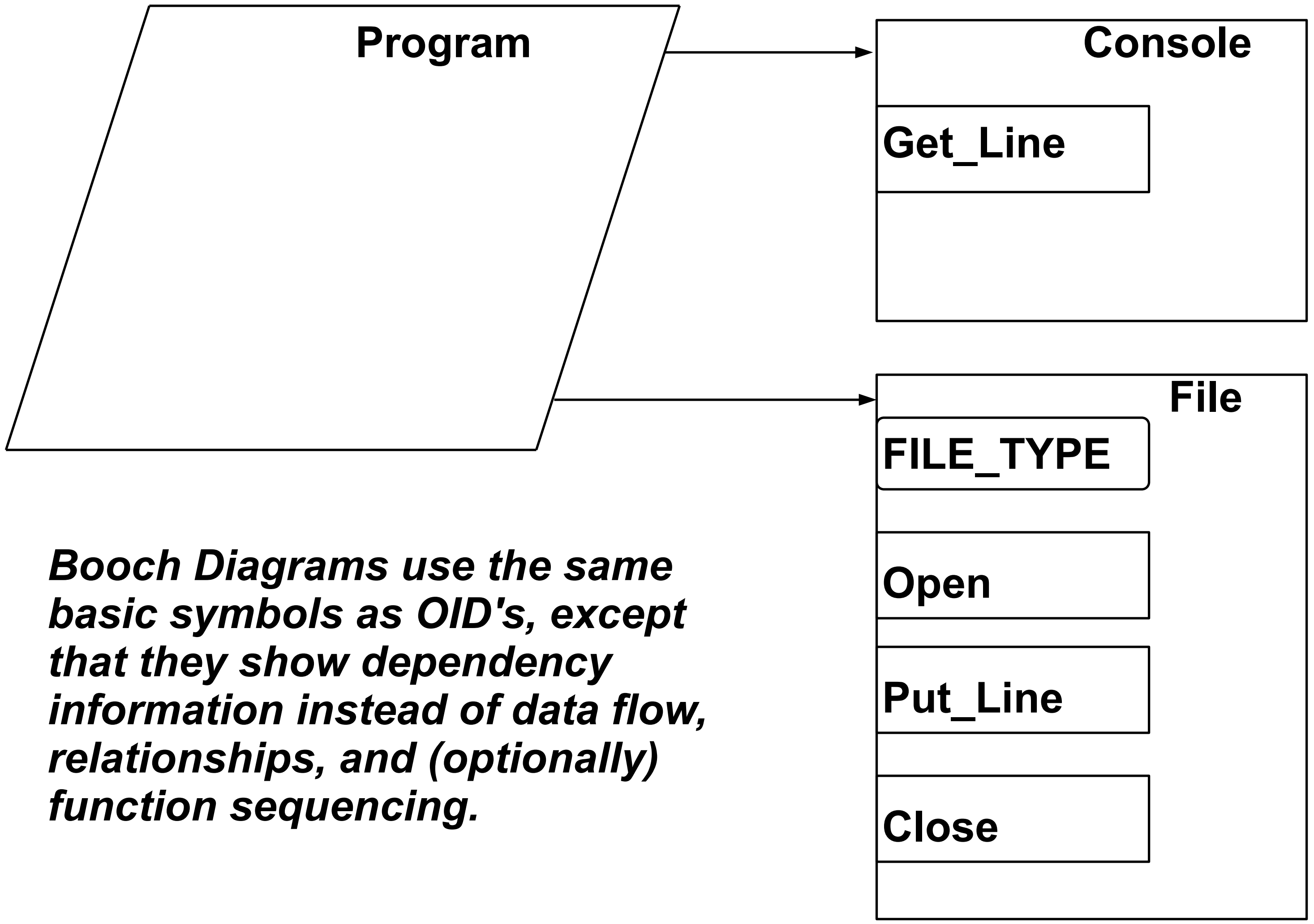


Invocation of one Subprogram by another with data flow

OIDs - Example



Booch Diagrams - Example



Booch Diagrams use the same basic symbols as OID's, except that they show dependency information instead of data flow, relationships, and (optionally) function sequencing.

Design Methodologies Real-Time Design

- **Data Flow-Oriented Design**
- **Data Structure-Oriented Design**
- **Object-Oriented Design**
- **Real-Time Design**

Note

The first three classes are heavily driven by the *Information Domain.*

Real-Time Design

- **Encompasses all aspects of conventional software design while simultaneously introducing timing and sizing constraints; these constraints must be satisfied by the code**
- **All classes of design (architectural, procedural, and data) become more complex due to the response time required by the real-world constraints**
- **Mathematical modeling and simulation are common tools used for real-time design**

Real-Time System Concerns

- **Interrupt handling and context switching**
- **Response time**
- **Data transfer rate**
- **CPU and system throughput**
- **Resource allocation and priority handling**
- **Task synchronization and intertask communication**