ICT 2402 Software Engineering

Software Design

Topics covered

- Software system design elements
  - Architectural design
  - Detailed design
  - Database design
  - User interface design

Software System Design

- Defines how software should be implemented by developers
- Main input to the design process is software requirements specification (SRS)
- Software design maps the SRS elements in to design elements

Design elements

- Architectural design
- Detailed design
- Database design
- User interface design

Architectural design

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design

- The output of this design process is a description of the software architecture

Cont…

- An early stage of the system design process
- Represents the link between specification and design processes
- Often carried out in parallel with some specification activities
- It involves identifying major system components and their communications

Advantages of explicit architecture

- 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

Architecture and system characteristics

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components
- Security
  - Use a layered architecture with critical assets in the inner layers
- Safety
  - Localize safety-critical features in a small number of sub-systems
- Availability
  - Include redundant components and mechanisms for fault tolerance
- Maintainability
  - Use fine-grain, replaceable components

Architectural conflicts

- Using large-grain components improves performance but reduces maintainability
- Introducing redundant data improves availability but makes security more difficult
- Localizing safety-related features usually means more communication so degraded performance

System structuring

- Concerned with decomposing the system into interacting sub-systems
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure
- More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed

Architectural design decisions

- Architectural design is a creative process so the process differs depending on the type of system being developed
- However, a number of common decisions span all design processes
  - Is there a generic application architecture that can be used?
  - How will the system be distributed?

Architectural design decisions

- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architecture reuse

- Systems in the same domain often have similar architectures that reflect domain concepts
  - E.g. Application product lines are built around a core architecture with variants that satisfy particular customer requirements

Architectural styles

- Architectural model of a system may conform to a generic architectural model or style
- An awareness of these styles can simplify the problem of defining system architectures
- Most large systems are heterogeneous and do not follow a single architectural style

Architectural models

- ***Static structural model*** that shows the major system components.
- ***Dynamic process model*** that shows the process structure of the system.
- ***Interface model*** that defines sub-system interfaces.
- ***Relationships model*** such as a data-flow model that shows sub-system relationships.
- ***Distribution model*** that shows how sub-systems are distributed across computers.

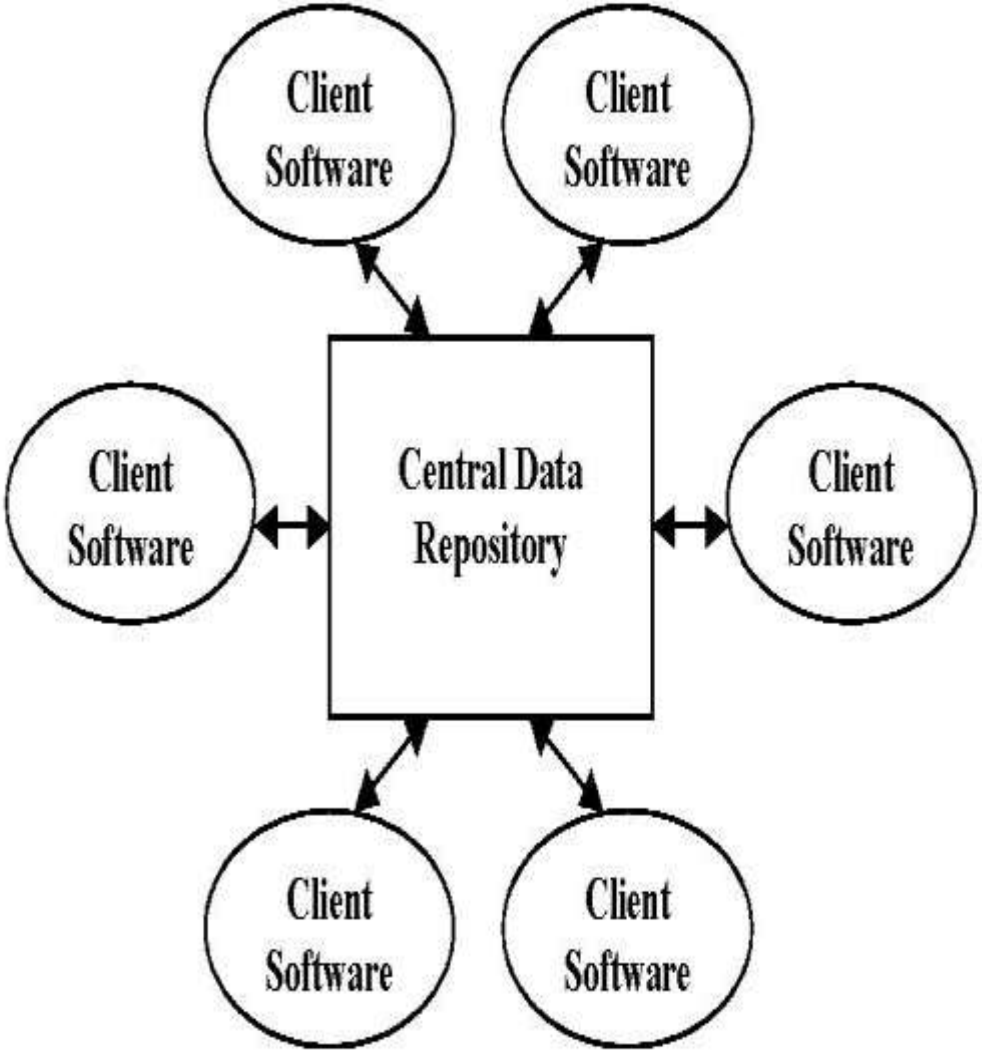
System organization models for architectural design

- Reflects the basic strategy that is used to structure a system
- Three organizational styles are widely used
    - Repository model
    - Client-server model
    - Layered model

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub systems
  - Each sub system maintains its own database and passes data explicitly to other sub systems
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

The repository model



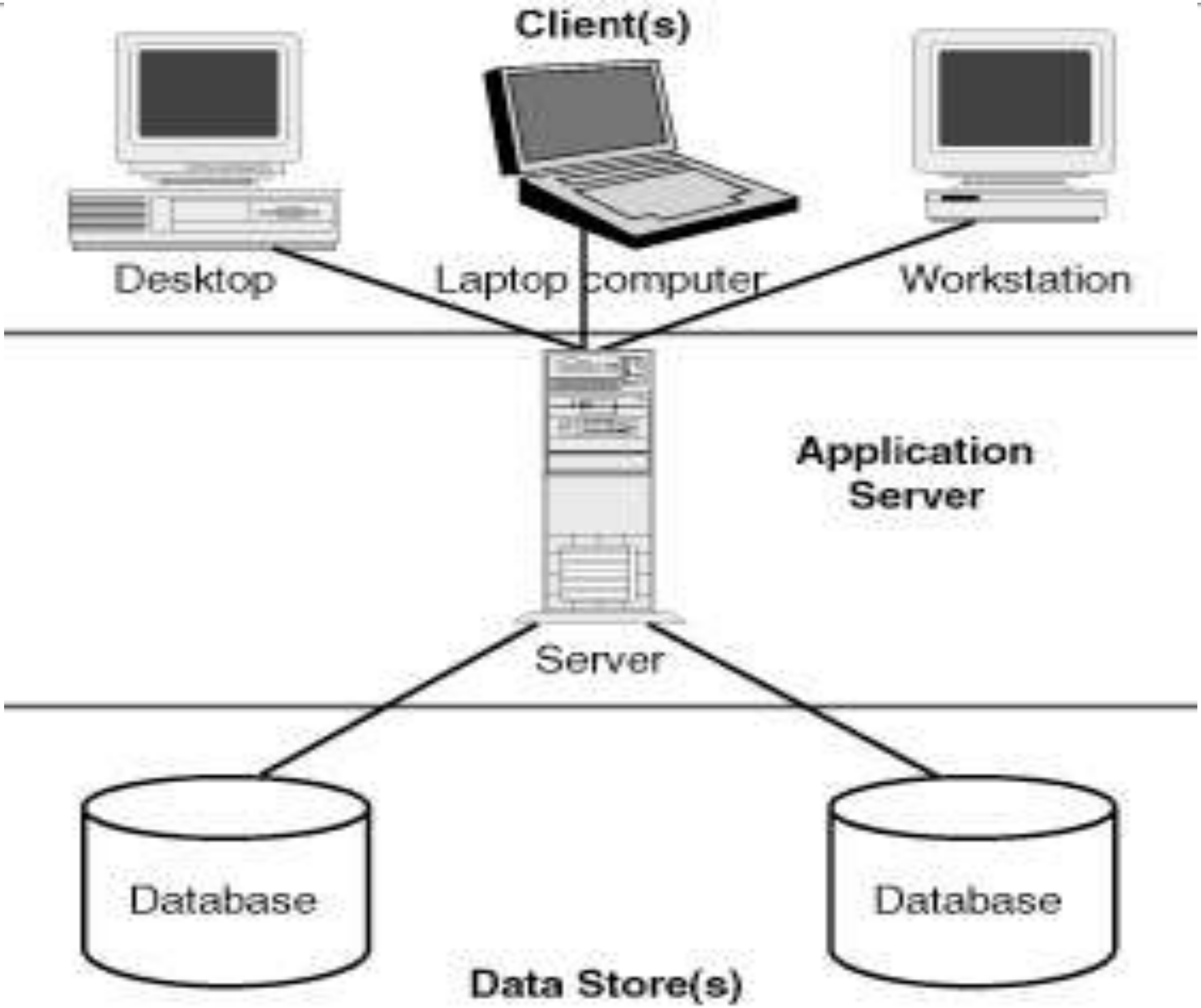
Repository model characteristics

- Advantages
  - Efficient way to share large amounts of data
  - Sub systems need not be concerned with how data is produced
  - Centralized management (backups, security etc.)
- Disadvantages
  - Sub systems must agree on a repository data model
  - Data evolution is difficult and expensive
  - No scope for specific management policies
  - Difficult to distribute efficiently

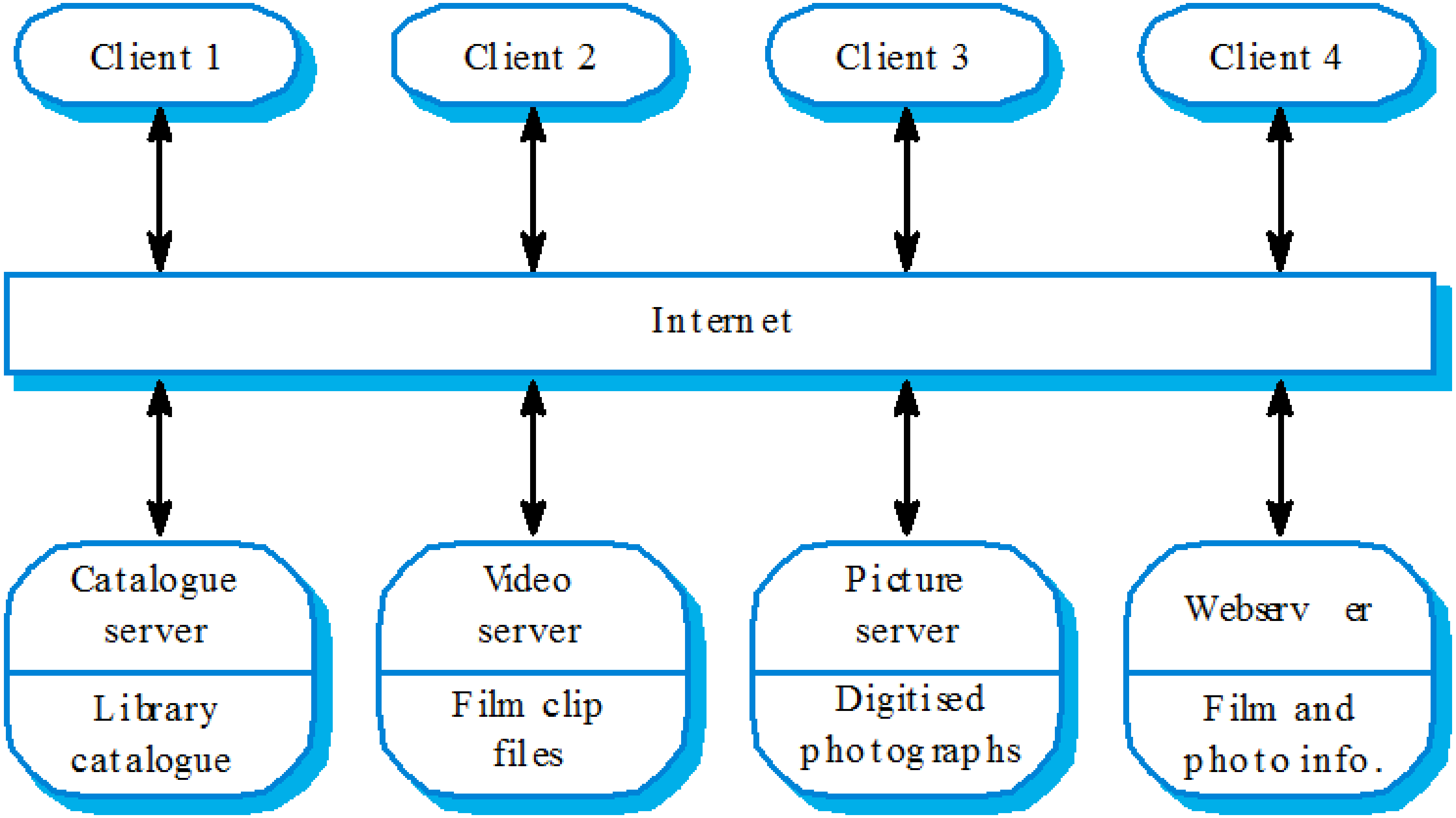
Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

Cont…


Client(s)
Desktop
Laptop computer
Workstation
Application Server
Server
Database
Database
Data Store(s)

Film and picture library



Client-server model characteristics

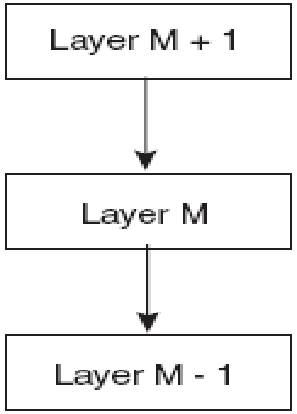
- Advantages
  - Data distribution is straight forward
  - Easy to maintain
  - Better security and control

- Disadvantages
  - Bandwidth limitations may exceed if the number of clients increased
  - In case of a server failure client functionality cannot be fulfilled
  - No central register of names and services. It may be hard to find out what servers and services are available.

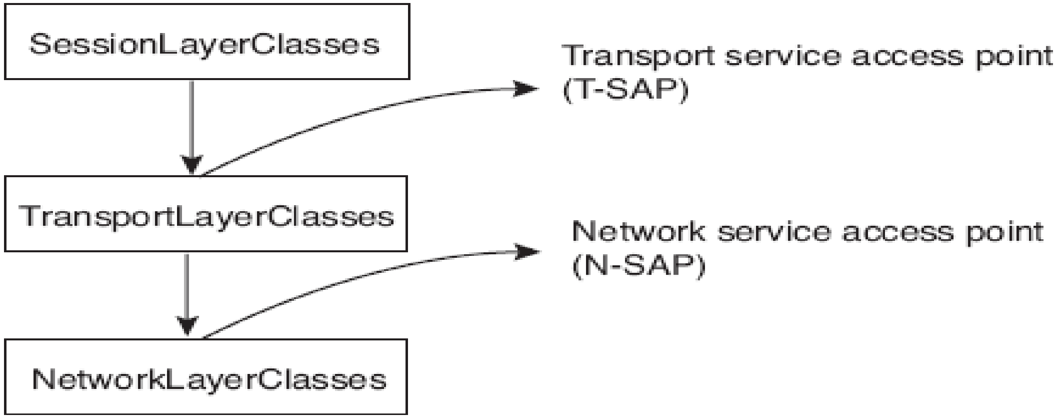
Layered model

- Used to model the interfacing of sub systems
- Organizes the system into a set of layers each of which provide a set of services
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

Cont...

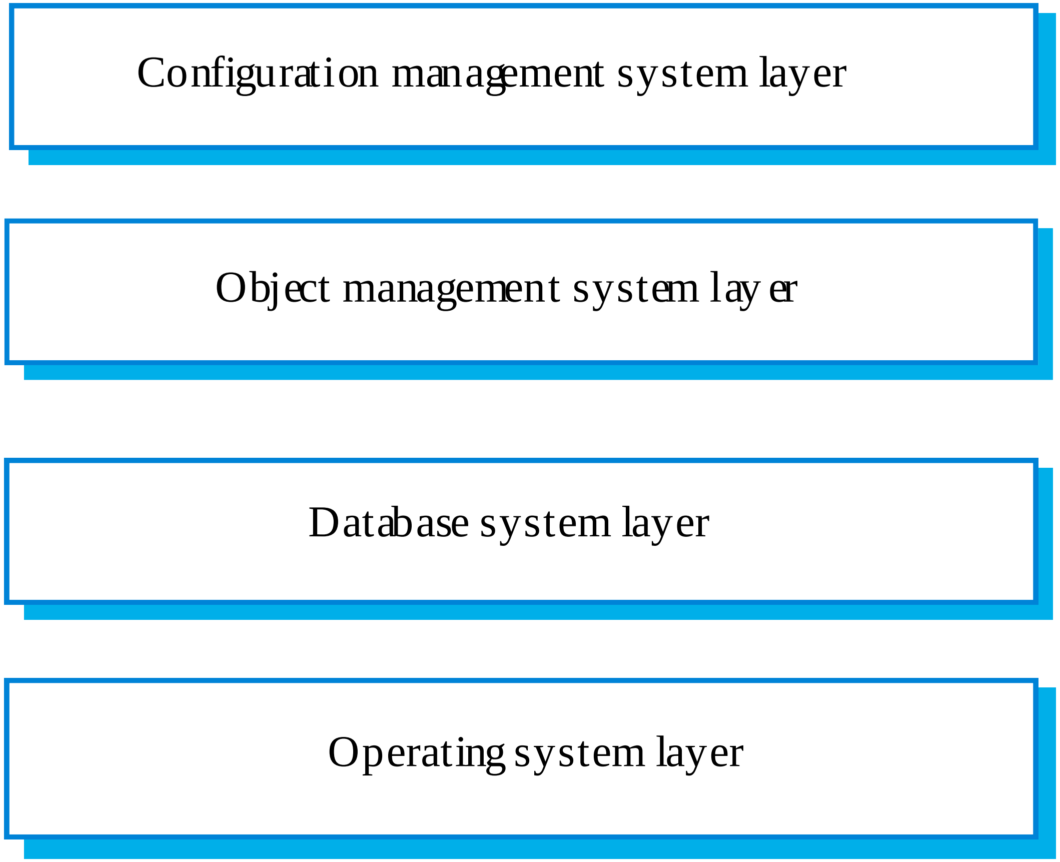


Layered architecture model



Part of the layered architecture of the OSI model

Version management system



Configuration management system layer

Object management system layer

Database system layer

Operating system layer

Modularity

- Modularity in design refers to the splitting of a large software system into smaller interconnected modules.
- Modules are interconnected through their interfaces. The interconnection should be as simple and little as possible to avoid side effects and costly maintenance.
- Two modules are **directly connected** if one module can call the other module.
- Two modules are **indirectly connected** if they share common files or global data structures.

Sub systems and modules

- A sub system is a system in its own right whose operation is independent of the services provided by other sub systems

- A module is a system component that provides services to other components but would not normally be considered as a separate system

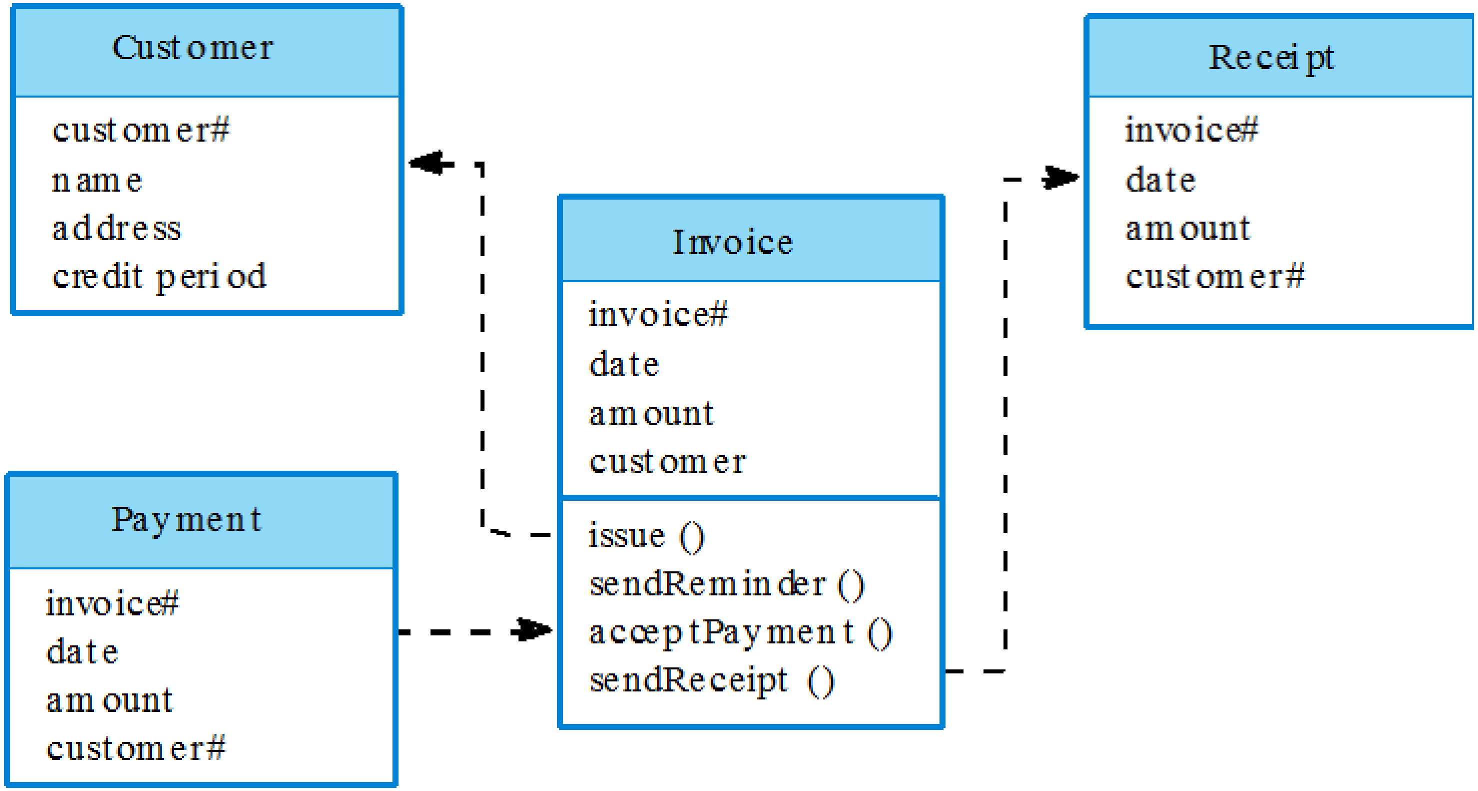
Modular decomposition

- Another structural level where sub-systems are decomposed into modules
- Two modular decomposition models used
  - An object model where the system is decomposed into interacting object
  - A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs
- If possible, decisions about concurrency should be delayed until modules are implemented

Object models

- Structure the system into a set of loosely coupled objects with well-defined interfaces
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations
- When implemented, objects are created from these classes and some control model used to coordinate object operations

Invoice processing system



Customer

customer#
name
address
credit period

Invoice

invoice#
date
amount
customer

issue ()
sendReminder ()
acceptPayment ()
sendReceipt ()

Receipt

invoice#
date
amount
customer#

Payment

invoice#
date
amount
customer#

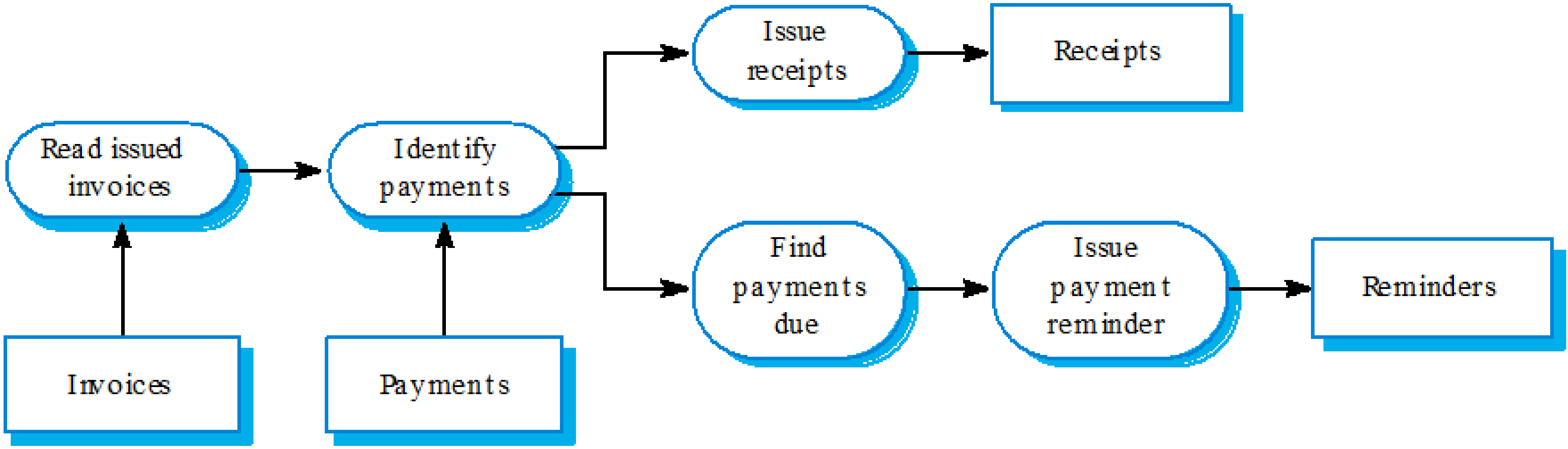
Object model advantages

- Objects are loosely coupled so their implementation can be modified without affecting other objects
- The objects may reflect real-world entities
- OO implementation languages are widely used
- *However, object interface changes may cause problems and complex entities may be hard to represent as objects*

Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell).
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive systems.

Invoice processing system



Pipeline model advantages

- Supports transformation reuse
- Intuitive organization for stakeholder communication
- Easy to add new transformations
- Relatively simple to implement as either a concurrent or sequential system
- *However, requires a common format for data transfer along the pipeline and difficult to support event-based interaction*

Cohesion

- A measure of how well a component “fits together” or focused.
- 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 localized in a single cohesive component.
- Various levels of cohesion have been identified.

Cohesion Levels

- Coincidental cohesion (weak)
  - Steps performed by the module are not related.
- Logical association (weak)
  - Components which perform similar functions are grouped.
- Temporal cohesion (weak)
  - Components which are activated at the same time are grouped.

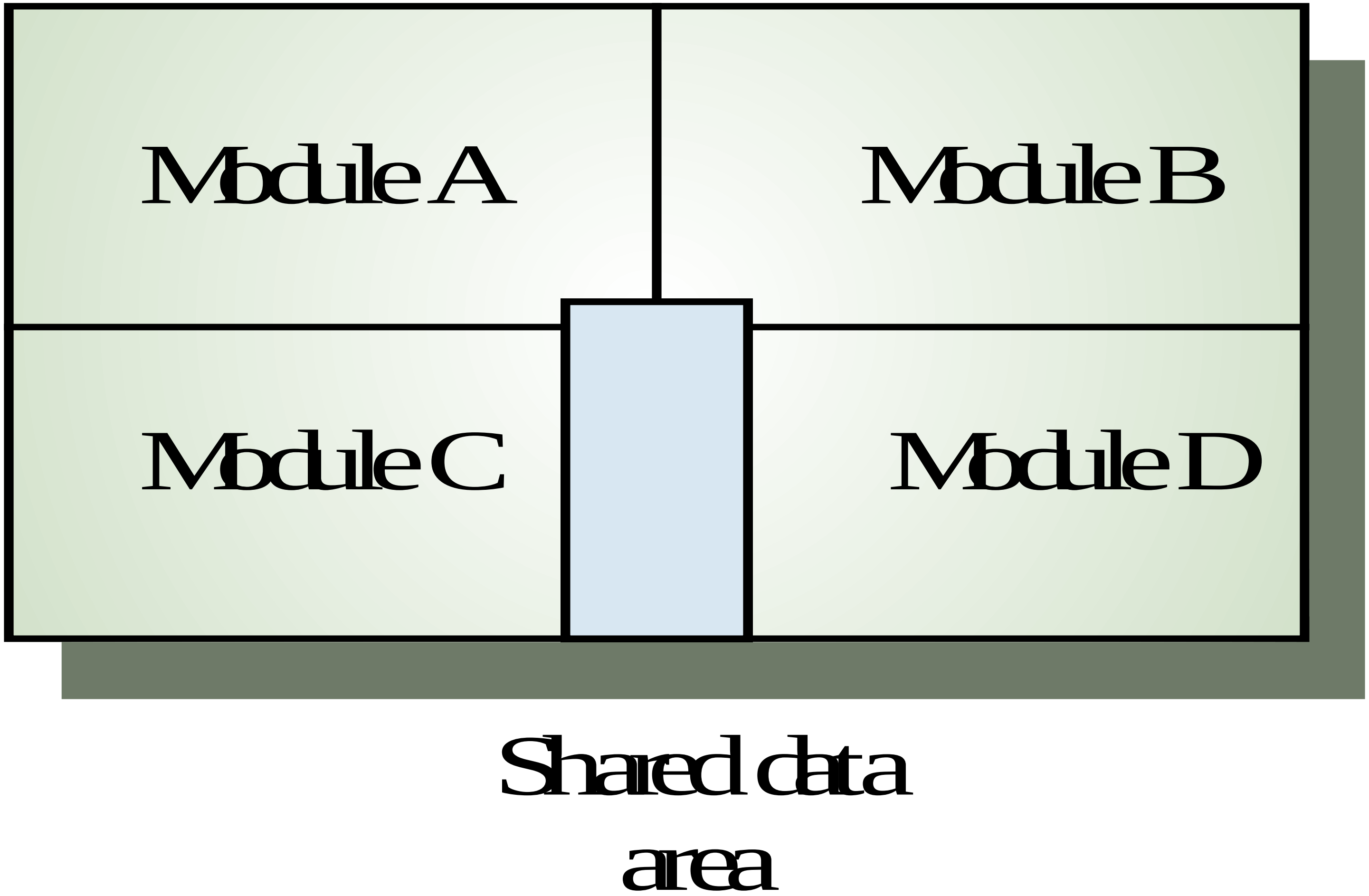
Cont...

- 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.

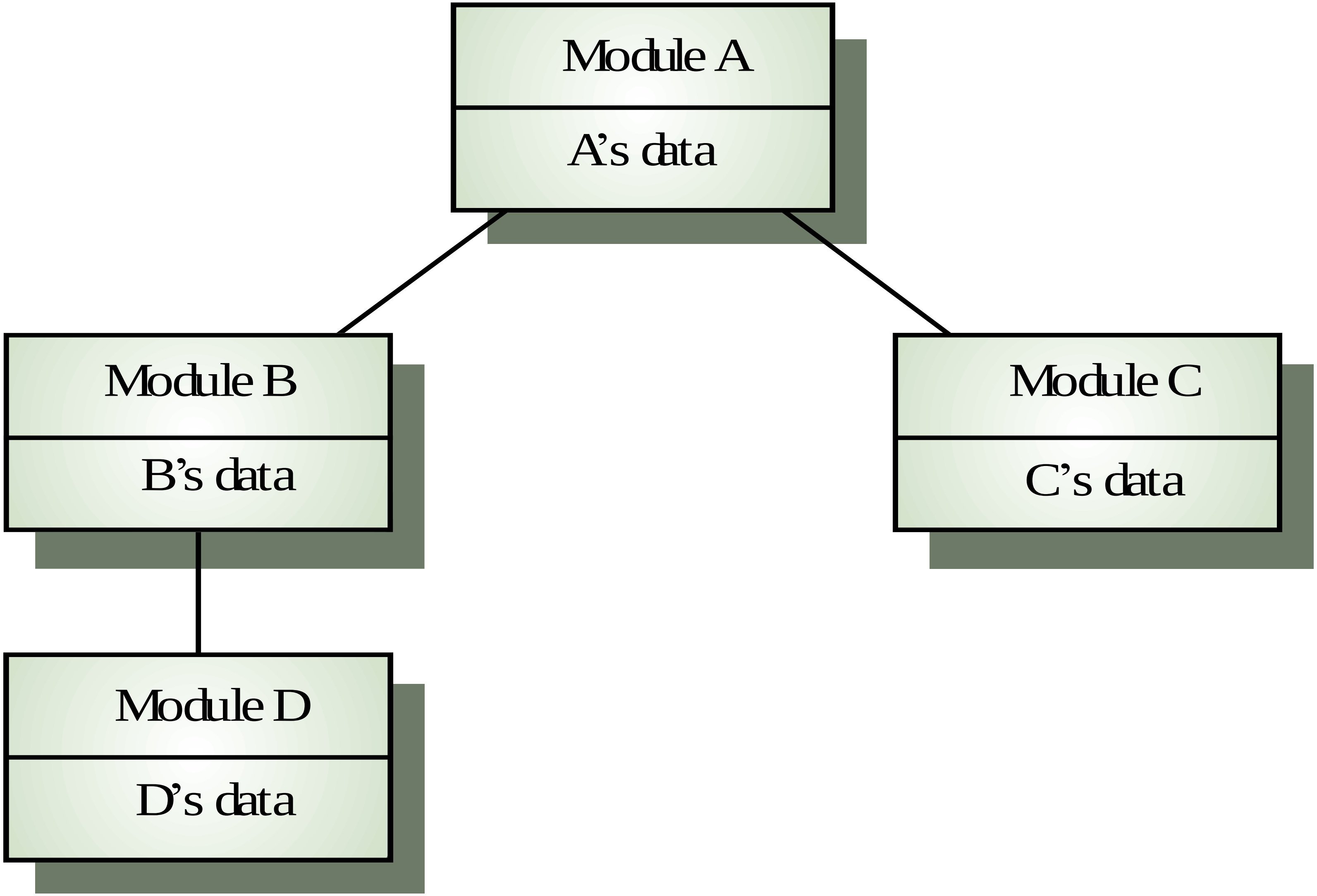
Coupling

- 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 decentralization (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.

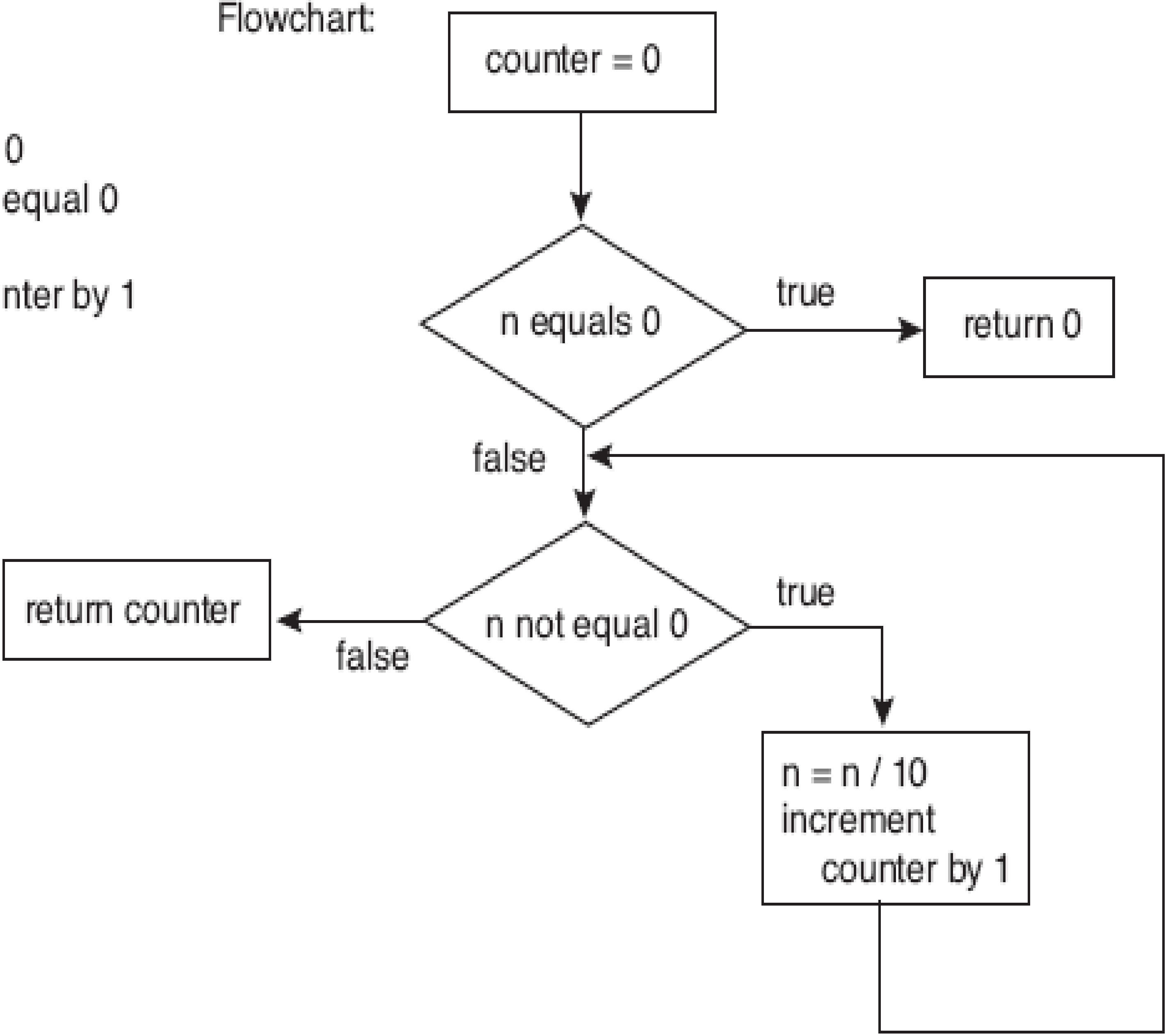
Detailed design

- Describing the module data (structures) and algorithm
- Several methods to specify detailed design,
  - Pseudo code
  - Flowchart
  - Activity diagrams
  - Templates

Pseudocode:

```
int counter = 0
if (n equals 0) return 0
repeat while n is not equal 0
        n = n / 10
        increment counter by 1
return counter
```

Flowchart:



counter = 0

n equals 0

true

return 0

false

n not equal 0

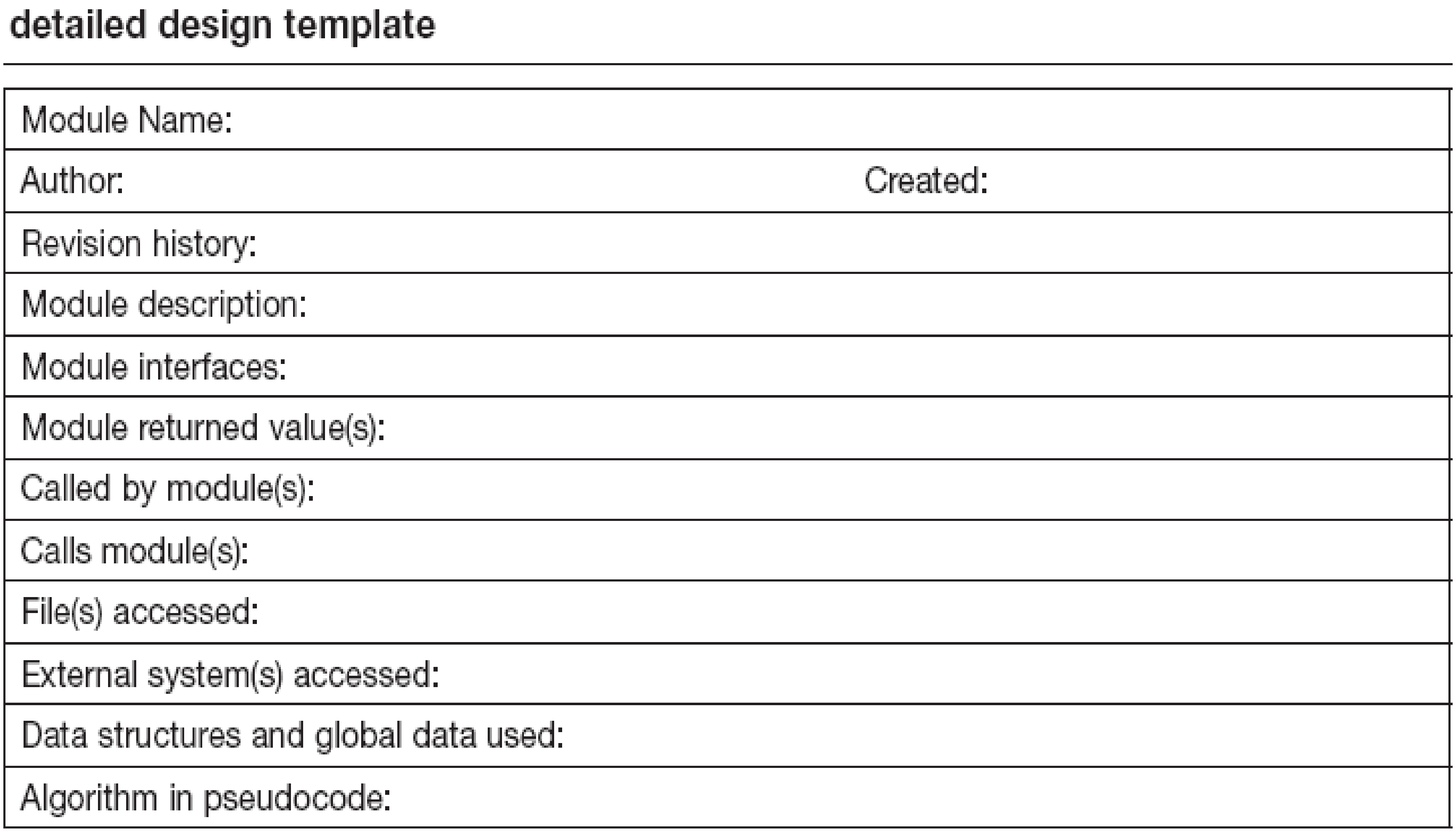
true

false

return counter

n = n / 10
increment
counter by 1

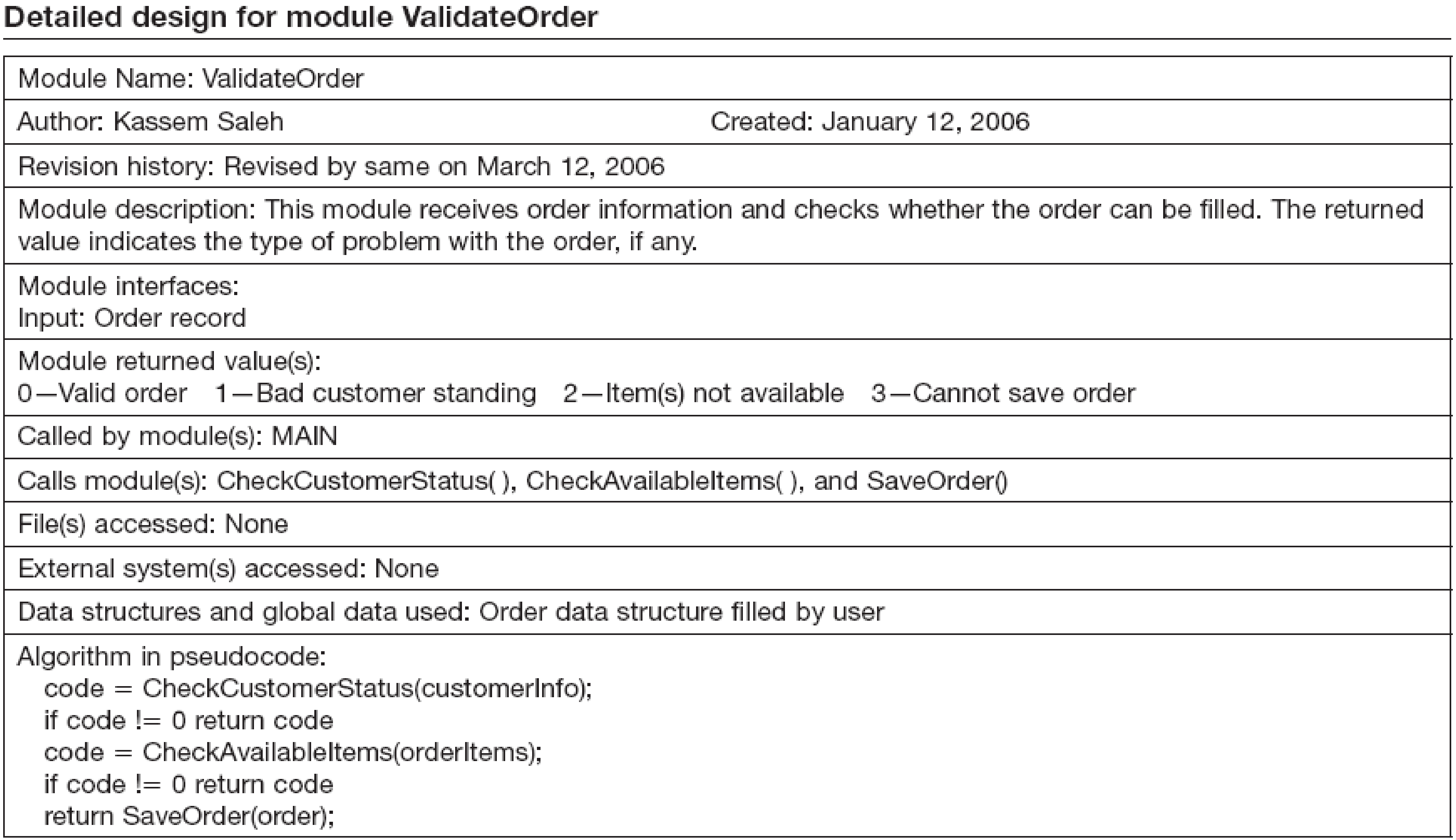
Module detailed design template



detailed design template

Module Name:	
Author:	Created:
Revision history:	
Module description:	
Module interfaces:	
Module returned value(s):	
Called by module(s):	
Calls module(s):	
File(s) accessed:	
External system(s) accessed:	
Data structures and global data used:	
Algorithm in pseudocode:	

Cont…



Detailed design for module ValidateOrder

Module Name: ValidateOrder

Author: Kassem Saleh Created: January 12, 2006

Revision history: Revised by same on March 12, 2006

Module description: This module receives order information and checks whether the order can be filled. The returned value indicates the type of problem with the order, if any.

Module interfaces:
Input: Order record

Module returned value(s):
0—Valid order 1—Bad customer standing 2—Item(s) not available 3—Cannot save order

Called by module(s): MAIN

Calls module(s): CheckCustomerStatus( ), CheckAvailableItems( ), and SaveOrder()

File(s) accessed: None

External system(s) accessed: None

Data structures and global data used: Order data structure filled by user

Algorithm in pseudocode:

```
   code = CheckCustomerStatus(customerInfo);
   if code != 0 return code
   code = CheckAvailableItems(orderItems);
   if code != 0 return code
   return SaveOrder(order);
```

Database design

- Prepare ER diagram and map it to database schema
- When mapping an ER diagram to a schema, two approaches may be followed
  - Normalization-schema is iteratively decomposed into relational schemas until all schemas reach at least the third normal form

  - Transformation-starts with the ER model and uses the appropriate transformation rules to obtain the various relation schemas needed to meet the data model requirements

Customer

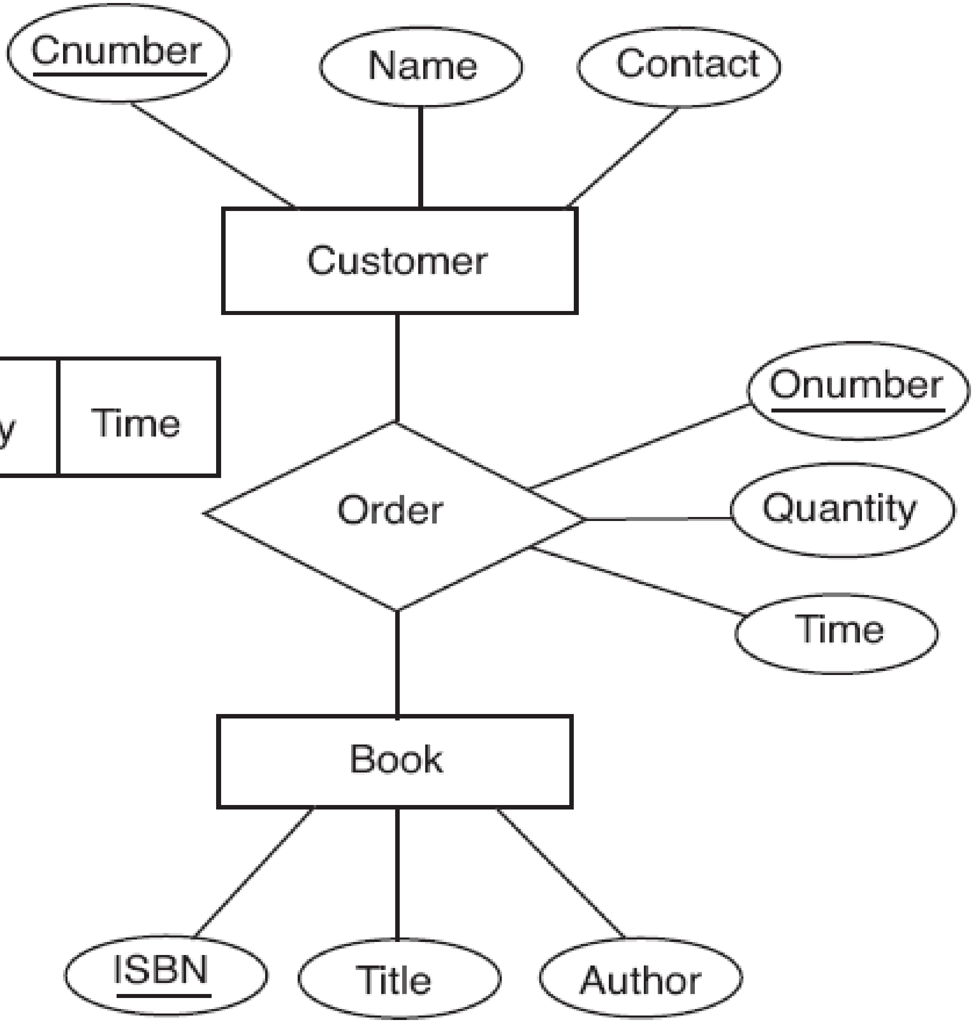
Cnumber	Name	Contact

Order

Onumber	CustomerNumber	ISBN	Qty	Time

Book

ISBN	Title	Author



Schema design for Customer, Order, and Book relations

Graphical User Interface Design

- Details on the interface specifications described in the SRS document
- Concern of the interface artifacts, their sequencing and interrelationships
- Larger projects may have a separate GUI development team for
  - Specification, design, implementation, testing
- Incorporate guidelines for designing GUI

Guidelines for good GUI

- Consistency
- Re-usability
- Flexibility and efficiency
- Forgiveness and tolerance
- Readability, simplicity and clarity
- User friendliness
- Visibility

Questions?