SWE 621:
Software Modeling and Architectural Design

Lecture Notes on Software Design
Lecture 1 - Introduction to Software Design

Hassan Gomaa

Dept of Computer Science
George Mason University
Fairfax, VA

Introduction to Software Design

1. Section I

Hassan Gomaa

References: H. Gomaa, "Chapters 1,2-5 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, 2000.

H. Gomaa, "Chapters 1-5 - H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures", Cambridge University Press, February 2011

Overview

- Follows general guidelines of Software Engineering Body of Knowledge (SWEBOK) – Chapter 3 Software Design
- Published by IEEE – 2004 Version
  - Fundamentals of Software Design
  - Software Design Process
  - Software Design Concepts
  - Software Design Notations and Methods

Software Design

What is design?

  noun: mental plan, preliminary sketch or outline

  verb: to conceive in the mind; to invent

What is software design?

  As a product

    Output of design process

  As a process

    Approach to doing design

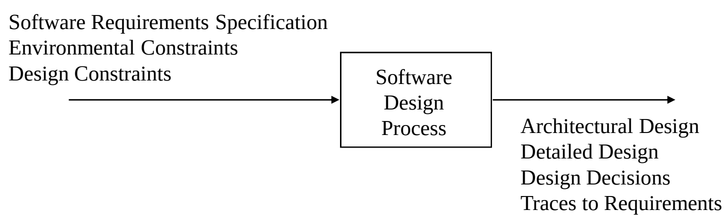
Nature of Design

- Design
  - Form of problem solving
- Design as “wicked problem”
  - Unlike an algorithm
    - There is no one “correct” solution
    - Tradeoffs in design
      - E.g., Structure vs. performance
      - Centralized vs. distributed
      - Sequential vs. concurrent

Software Design Activities

- Architectural Design
  - Structure system into components
  - Define the interfaces between components
- Detailed Design
  - Define internal logic
  - Define internal data structures

Context of Software Design



Inputs To Software Design

- Software requirements specification
  - Describes WHAT system shall do not HOW
  - External view of system to be developed
- Environmental constraints
  - Hardware, language, system usage
- Design constraints
  - Design method
  - Design notation

Outputs From Software Design

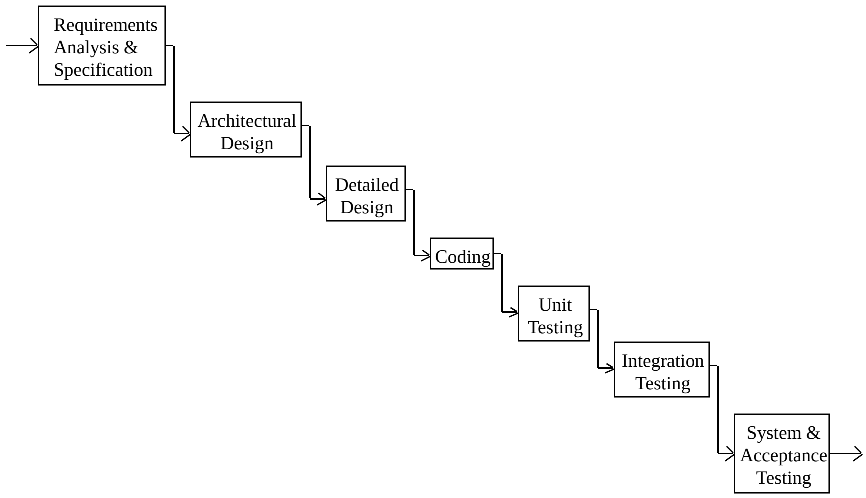
- Architectural Design
  - Overall description of software structure
    - Textual and Graphical
  - Specification of software components and their interfaces
    - Modules, classes
- Detailed Design of each component
  - Internal logic
  - Internal data structures
- Design decisions made
  - Design rationale
- Traces to requirements

Software Design Process

- Software life cycle (a.k.a. software process)
  - Phased approach to software development
- Software life cycle (a.k.a. process) models
  - Waterfall – limitations of Waterfall Model
  - Incremental - evolutionary prototyping
  - Exploratory - throwaway prototyping
  - Spiral model – risk driven process model

Software Life Cycle

Waterfall Model



Software Life Cycle Model
Software Definition

- Requirements Analysis and Specification
  - Analysis of user's problem
  - Specification of "what" system shall provide user
- Architectural Design
  - Specification of "how" system shall be structured into components
  - Specification of interfaces between components

Software Life Cycle Model
Software Construction

- Detailed Design
  - Internal design of individual components
    - Design of logic and data structures
- Coding
  - Map component design to code
- Unit Testing
  - Test individual components

Software Life Cycle Model
Software Integration and Test

- Integration Testing
  - Gradually combine components and test combinations
- System Testing
  - Test of entire system against software requirements
- Acceptance Test
  - Test of entire system by user prior to acceptance

Software Life Cycle Model
Software Maintenance

- Modification of software system after installation and acceptance
  - Fix software errors
  - Improve performance
  - Address changes in user requirements
- Often implies significant software redesign

Limitations of Waterfall Model

- Does not show iteration in software life cycle
- Does not show overlap between phases
- Software requirements are tested late in life cycle
- Operational system available late in life cycle

Prototyping During Requirements Phase

Problem

- Software requirements are tested late in life cycle

Solution

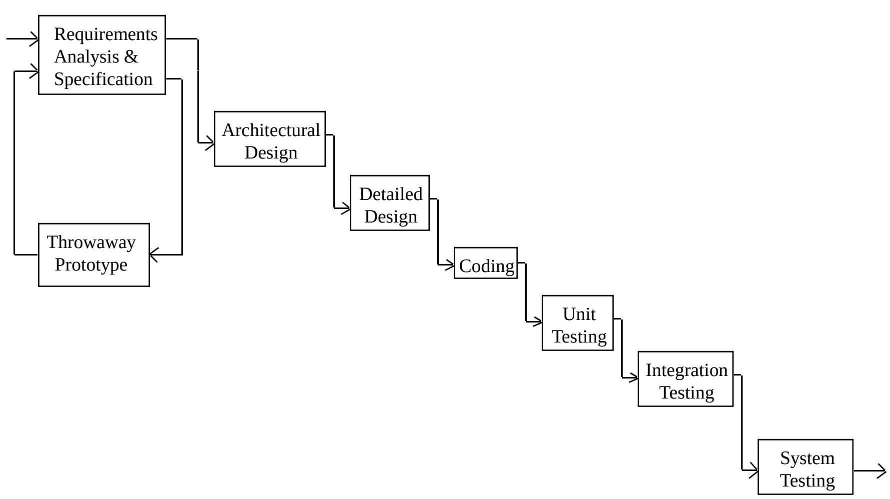
- Use throw-away prototyping

Help ensure requirements are understood

Also first attempt at designing system

- Design of key file and data structures
- Design of user interface
- Early design tradeoffs

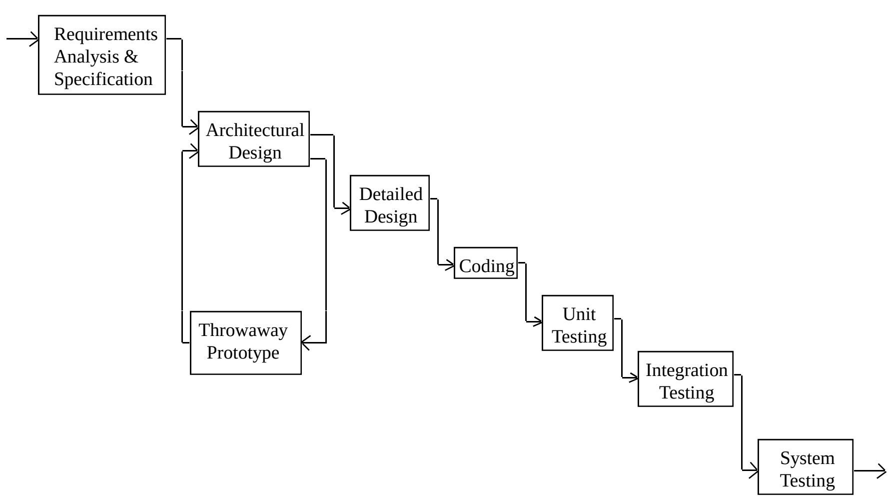
Impact of Throwaway Prototyping on Software Life Cycle



Throw-away Prototyping in Design

- Objectives
  - Test design early
  - Experiment with alternative design decisions
- Examples of prototyping in design
  - Algorithm design
    - Experiment with - speed, accuracy
  - Early performance analysis
    - Measure timing parameters
  - User interface

Impact of Throwaway Prototyping on Architectural Design Phase



Incremental Development

Problem

- Operational system available late in life cycle

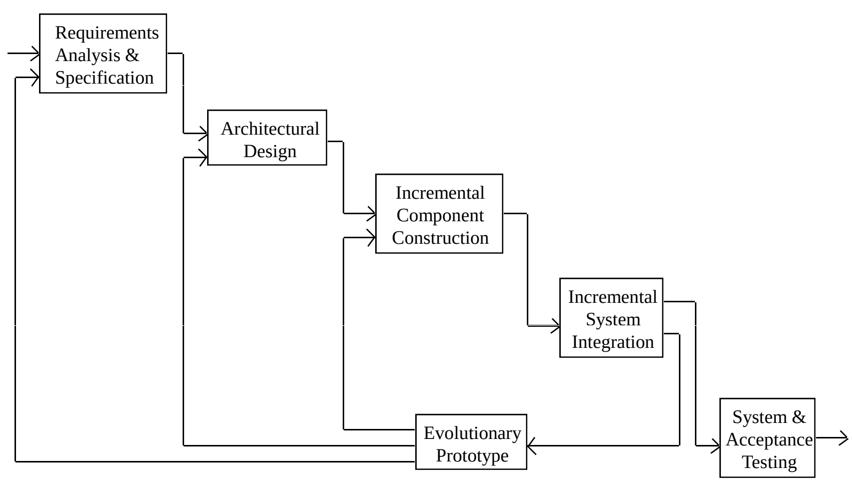
Solution

- Use incremental development
- Also known as evolutionary prototyping

Objective

- Subset of system working early
- Gradually build on
- Prototype evolves into production system

Incremental Development Software Life Cycle



Should Prototype Evolve into Production System?

Tradeoff

- Rapid development
- Quality of product

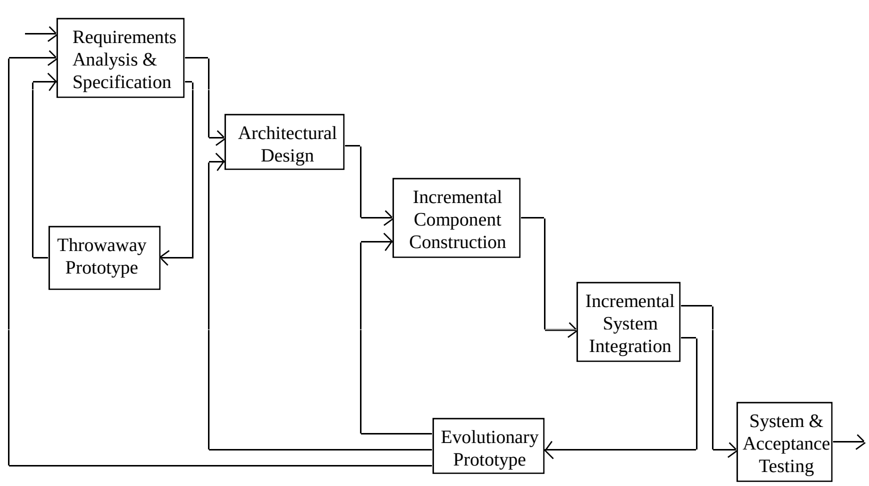
Throw-away prototype

- Speed, not quality is goal
- Must not evolve into production system

Evolutionary prototype

- Must emphasize quality
- Maintainability is key issue

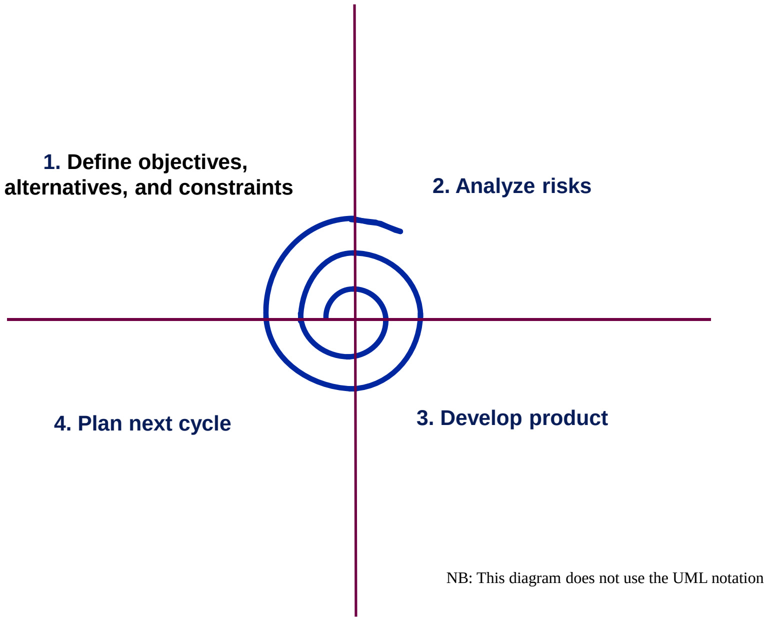
Combined Throwaway Prototyping / Incremental Development Software Life Cycle Model



Spiral Process Model (SPM)

- SPM consists of four main activities that are repeated for each cycle (Fig. 5.6):
  - Defining objectives, alternatives and constraints
  - Analyzing risks
  - Developing and verifying product
  - Spiral planning
- Number of cycles is project specific
- Risk driven process
  - Analyze risks in second quadrant

Figure 5.6 The spiral process model



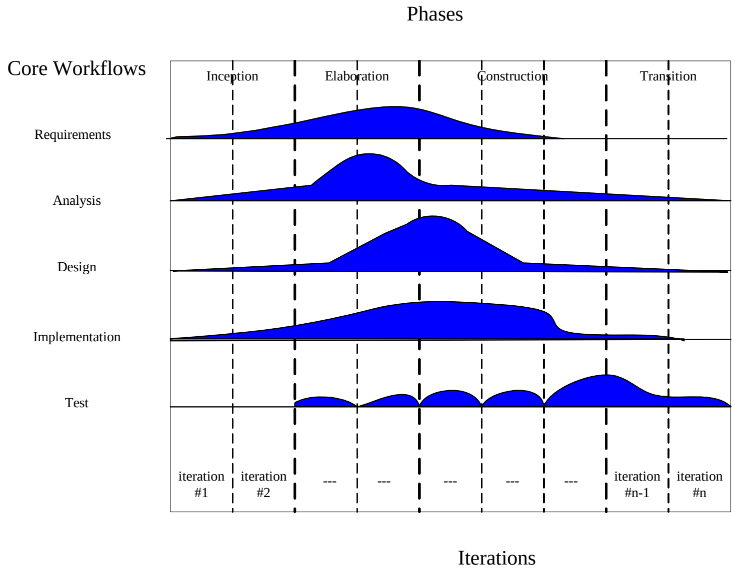
Unified Software Development Process

- Risk driven iterative process
  - Also known as *Rational Unified Process*
- Workflow
  - Sequence of activities that produces a result of observable value
- Workflows in Unified Process
  - **Requirements**
    - Product: Use case model.
  - **Analysis**
    - Product: Analysis model.
  - **Design**
    - Products: design model and deployment model.
  - **Implementation**
    - Product: software implementation
  - **Test**.
    - Products: Test cases and test results

Unified Software Development Process

- **Phase**
  - Time between two major milestones
- Phases in Unified Process
  - **Inception**
    - Seed idea is developed
  - **Elaboration**.
    - Software architecture is defined
  - **Construction**.
    - Software is built to the point at which it is ready for release
  - **Transition**.
    - Software is turned over to the user community.

Figure 3.5: Unified Software Development Process



Software Design Concepts

- Objects and Classes
- Information Hiding
- Inheritance
- Concurrency
- Finite State Machines

Objects and Classes

- Objects represent “things” in real world
  - Provide understanding of real world
  - Form basis for a computer solution
- An Object (object instance) is a single “thing”
  - E.g., John’s car
  - Mary’s account
- A Class (object class) is a collection of objects with the same characteristics
  - E.g., account, employee, car, customer
- **Figure 2.2 UML notation for objects & classes**
- **Figure 3.1 Example of classes and objects**

Figure 2.2 UML notation for objects & classes

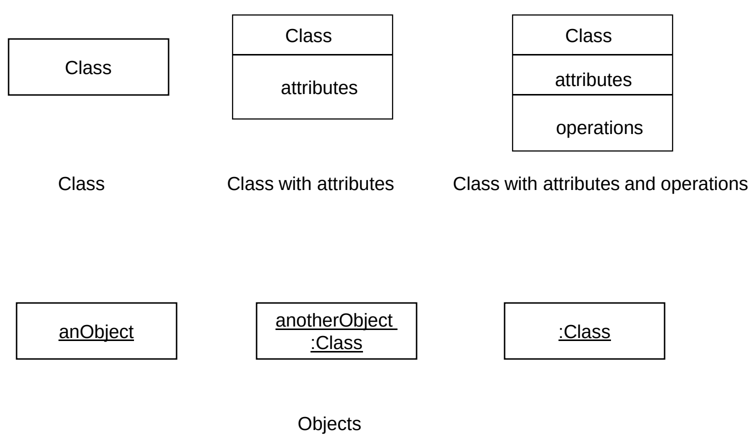
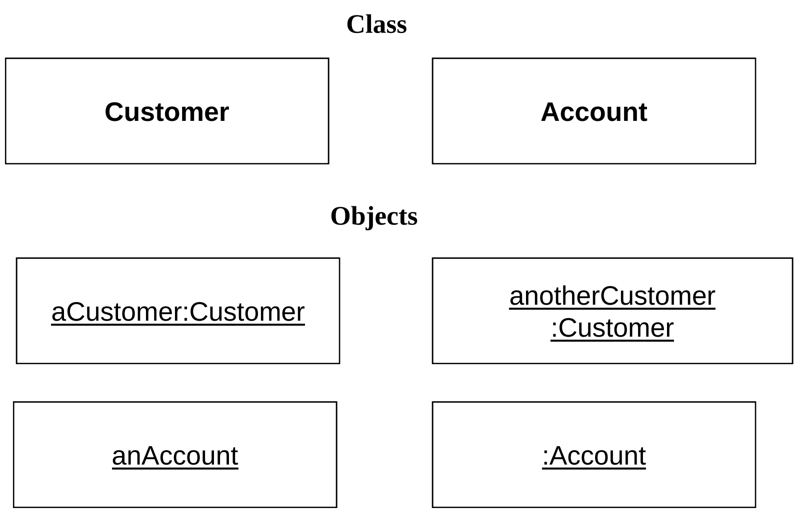


Figure 3.1 Example of classes and objects

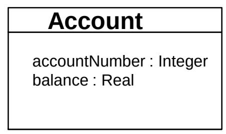


Attributes

- Attribute
  - Data value held by object in class
- Example of Attributes
  - E.g., account number, balance
- Each object instance has specific value of attribute
  - John's account number is 1234
  - Mary's account number is 5678
- Attribute name is unique within class
- **Figure 3.2 Example of class with attributes**

Figure 3.2 Example of class with attributes

Class with attributes

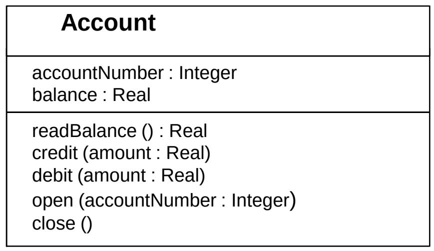


Objects with values

Classes and Operations

- Operation
  - Is function or procedure that may be applied to objects in a class
  - All objects in class have same operations
- Class has one or more operations
  - Operations manipulate values of attributes maintained by object
- Operations may have
  - Input parameters
  - Output parameters
  - Return value
- Signature of operation
  - Operation's name
  - Operation's parameters
  - Operation's return value
- Interface of class
  - Set of operations provided by class
- **Figure 3.3 Class with attributes and operations**

Figure 3.3 Class with attributes and operations



Information Hiding

- Each object hides design decision
  - E.g., data structure
    - interface to I/O device
- Information hiding object
  - Hides (encapsulates) information
  - Accessed by operations
- Basis for Object-Oriented Design
- Advantage
  - Objects are more self-contained
  - Results in more modifiable -> maintainable system

Example of Information Hiding

- Example of Stack
- Conventional approach
  - Stack data structure is global
  - Stack accessed by modules
  - Module corresponds to procedure / function / subroutine
- – Problem
  - Change to stack data structure has global impact
- Consider
  - Array implementation (Fig. 3.4) changed to
  - Linked list implementation (Fig. 3.6)
- Every module is impacted by change

Figure 3.4 Example of Global Access to Data

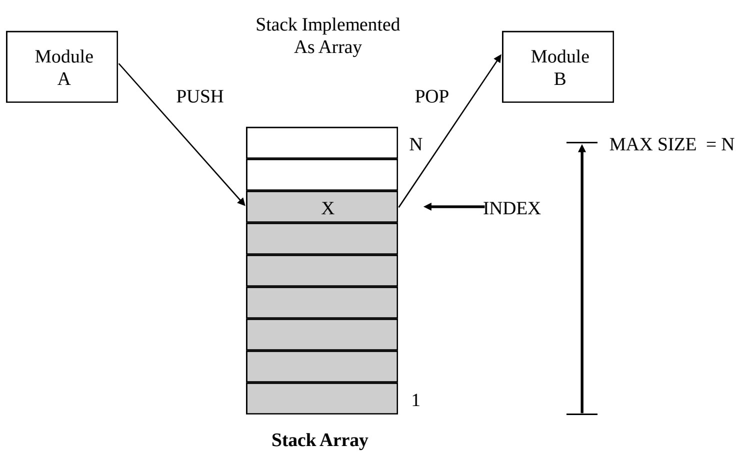
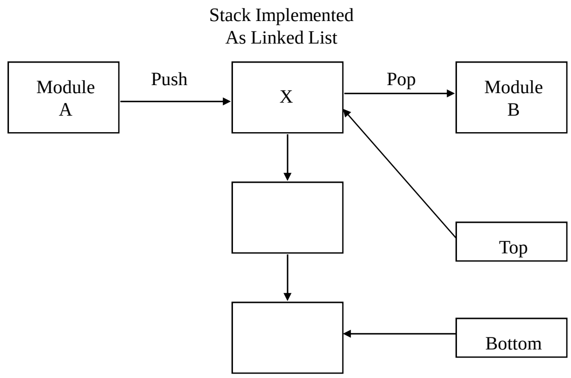


Figure 3.6 Example of Global Access to Data



Example of Information Hiding

- Example of Stack
- Information hiding solution
  - Hide stack data structure and internal linkage
  - Specify operations on stack data structure
  - Access to stack only via operations
- Consider
  - Array implementation (Fig. 3.5) changed to
  - Linked list implementation (Fig. 3.7)
- Change to stack only impacts Stack object

Figure 3.5 Example of Information Hiding

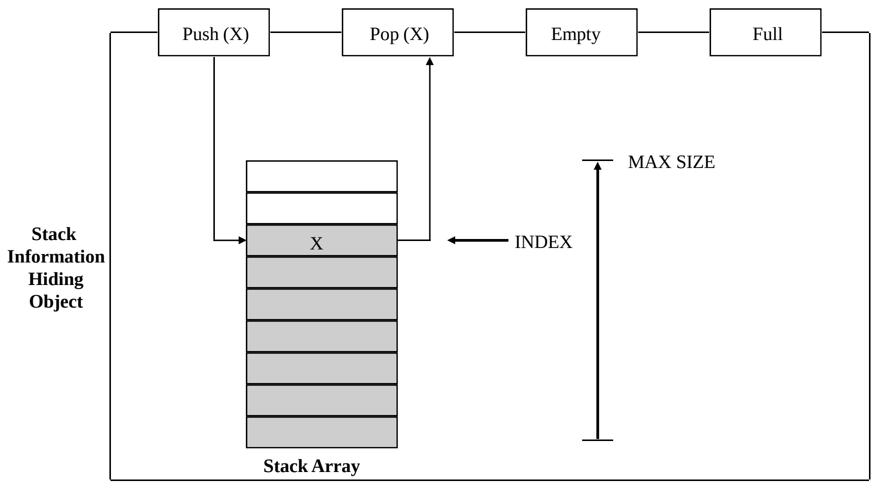
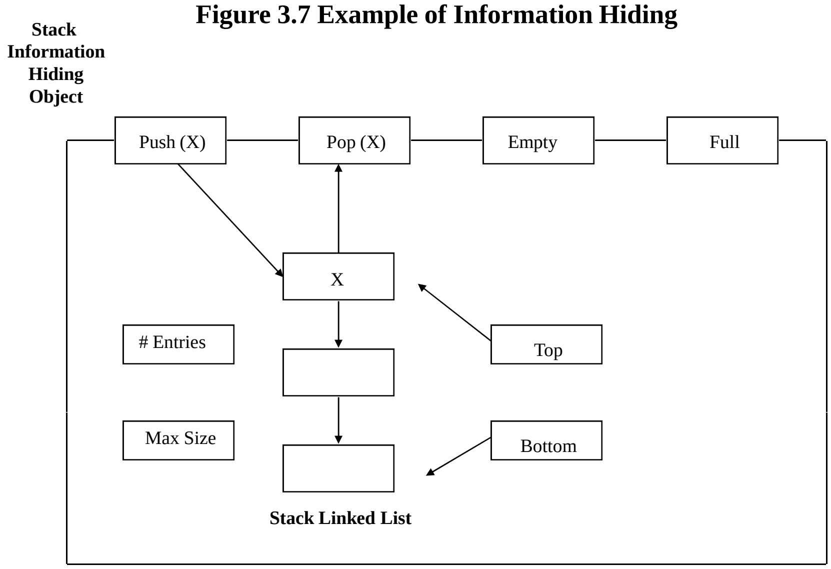


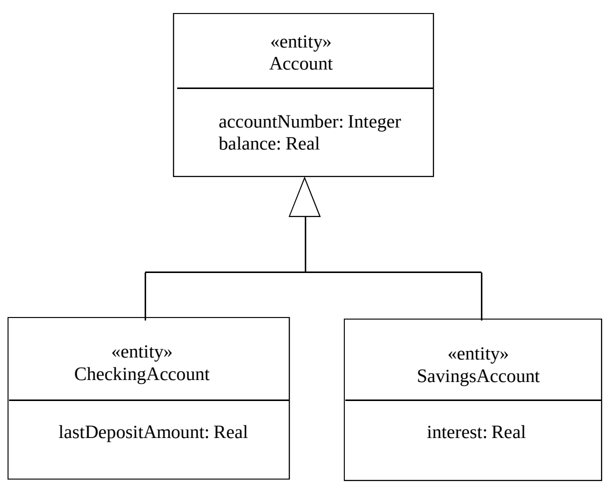
Figure 3.7 Example of Information Hiding



Inheritance in Design

- Subclass inherits generalized properties from superclass
- Inheritance
  - Allows sharing of properties between classes
    - Property is Attribute or Operation
  - Allows adaptation of parent class (superclass) to form child class (subclass)
- Subclass inherits attributes & operations from superclass
  - May add attributes
  - May add operations
  - May redefine operations

Generalization / specialization hierarchy



Sequential & Concurrent Problems

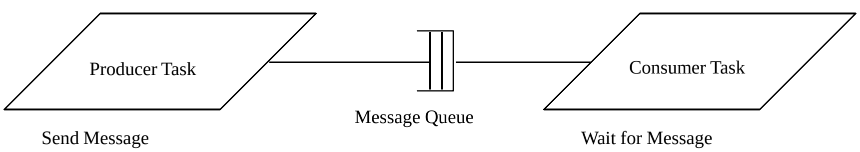
- Sequential problems
  - Activities happen in strict sequence
    - E.g., compiler, payroll
  - Sequential solution = program
- Concurrent problems
  - Many activities happen in parallel
    - E.g., multi-user interactive system, air traffic control system
  - Sequential solution to concurrent problem increases design complexity

Concurrent and Real-Time Systems

- Concurrent System
  – Consists of many activities (tasks) that execute in parallel
- Real-Time system
  – Concurrent system with timing deadlines
- Distributed application
  – Concurrent system executing on geographically distributed nodes

Concurrency

- Characteristics of concurrent task
  - A.k.a. (lightweight) process, thread
    - Active object, concurrent object
  - One sequential thread of execution
  - Represents execution of
    - Sequential program
    - Sequential part of concurrent program
  - Concurrent system
    - Many tasks execute in parallel
    - Tasks need to interact with each other

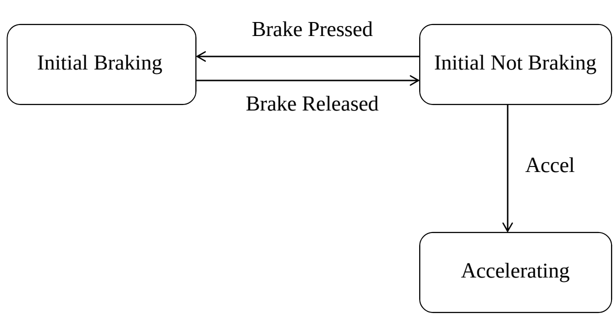


Asynchronous Message Communication between Concurrent Tasks

Finite State Machines

- Many information and real-time systems are state dependent
  - Action depends not only on input event
  - Also depends on state of system
- Finite State Machine
  - Finite number of states
  - Only in one state at a time
- State
  - A recognizable situation
  - Exists over an interval of time
- Event
  - A discrete signal that happens at a point in time
  - Causes change of state

Figure 10.4 Partial statechart



Software Design Terminology

- Design concept or principle
  - Fundamental idea that can be applied to designing a system, e.g., information hiding
- Design notation or representation
  - A means of describing a software design
    - Textual and Graphical, e.g., UML
- Design strategy
  - Overall plan and direction for performing design
- Design structuring criteria
  - Guidelines for decomposing a system into its parts

Software Design Method

- Systematic approach for creating a design
  - Design decisions to be made
  - Order in which to make them
- Describes sequence of steps for producing a design
  - Based on set of design concepts
  - Employs design strategy(ies)
  - Provides design structuring criteria
  - Documents resulting design using design notation(s)

Example of Software Design Method
Structured Design

Design concept
- Functional module

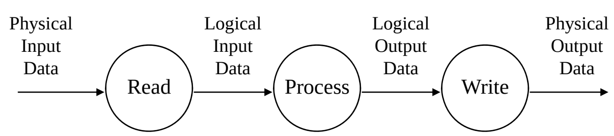
Design structuring criteria
- Module Cohesion criteria
  - Unity within module
- Module Coupling criteria
  - Connectivity between modules

Design strategy
- Transaction Analysis and Transform Analysis

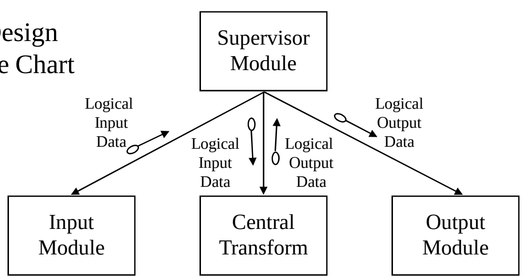
Design notation
- Structure charts
- Program Design Language (PDL)

Design Strategies
Transform Analysis

- Structured Analysis
  - Data flow diagram



- Structured Design
  - Structure Chart



Example of Software Design Method COMET

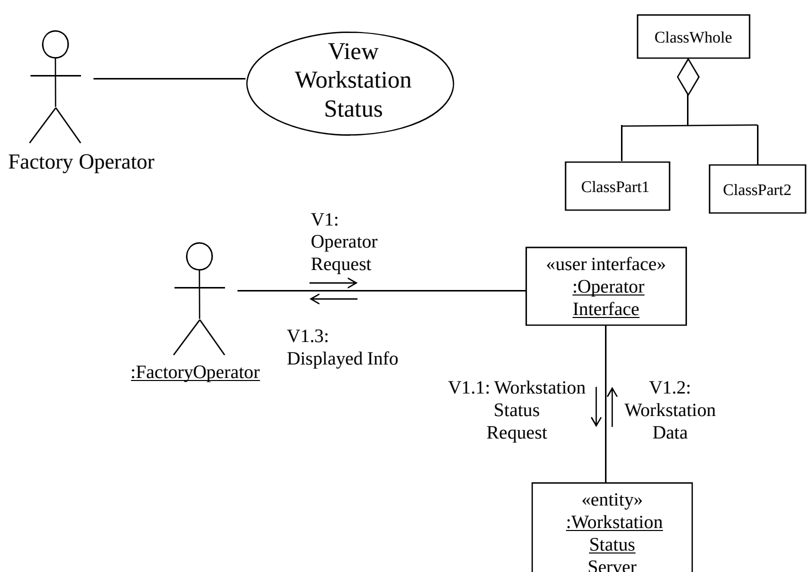
Design concepts
- Finite state machine, concurrent task, information hiding

Design structuring criteria
- Object, subsystem and task structuring criteria

Design strategy
- Develop analysis model, then map to design model

Design notation
- UML (Unified Modeling Language)

Example of Software Design Method COMET



Review

- Follows general guidelines of Software Engineering Body of Knowledge (SWEBOK) – Chapter 3 Software Design
- Published by IEEE – 2004 Version
  - Fundamentals of Software Design
  - Software Design Process
  - Software Design Concepts
  - Software Design Notations and Methods