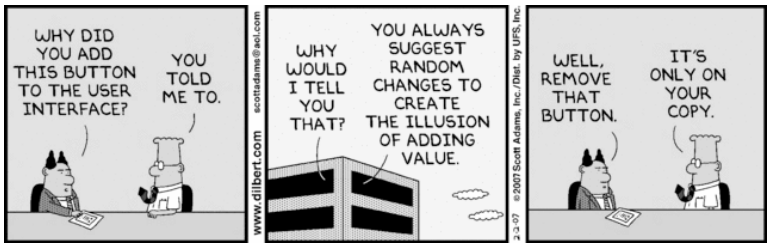
Designing the Module Structure

How do we design to arrive at desired qualities?

Address Book exercise



Architecture Design Process

Building architecture to address business goals:

1. Understand the goals for the system
2. Define the quality requirements
3. *Design the architecture*
   1. Views: which architectural structures should we use? (goals<->architectural structures<->representation)
   2. Documentation: how do we communicate design decisions?
   3. Design: how do we decompose the system?
4. Evaluate the architecture (is it a good design?)

Which structures should we use?

Structure	Components	Interfaces	Relationships
Calls Structure	Programs (methods, services)	Program interface and parameter declarations	Invokes with parameters (A calls B)
Data Flow	Functional tasks	Data types or structures	Sends-data-to
Process	Sequential program (process, thread, task)	Scheduling and synchronization constraints	Runs-concurrently-with, excludes, precedes

- Choice of structure depends the *specific design goals*
  - Compare to architectural blueprints
- Choose minimal set of structures that
  - Make key design issues visible
  - Communicate key design decisions
- Which views would be useful for Address Book?

Important project qualities?

Behavioral (observable)

- Performance
- Security
- Availability
- Reliability
- Usability

Properties resulting from the properties of components, connectors and interfaces that exist at run time.

Developmental Qualities

- Modifiability(ease of change)
- Portability
- Reusability
- Ease of integration
- Understandability
- Extensibility (extend/contract)
- Provide independent work assignments

Properties resulting from the properties components, connectors and interfaces that exist at design time *whether or not they have any distinct run-time manifestation*.

Some Key Architectural Structures

- Module Structure*
  - Decomposition of the system into work assignments or information hiding modules
  - Most influential design time structure
    - Modifiability, work assignments, maintainability, reusability, understandability, etc.
- Uses Structure
  - Determine which modules may use one another's services
  - Determines subsetability, ease of integration (e.g. for increments)
- Process Structure
  - Decomposition of the runtime code into threads of control
  - Determines potential concurrency, real-time behavior

The Module Structure

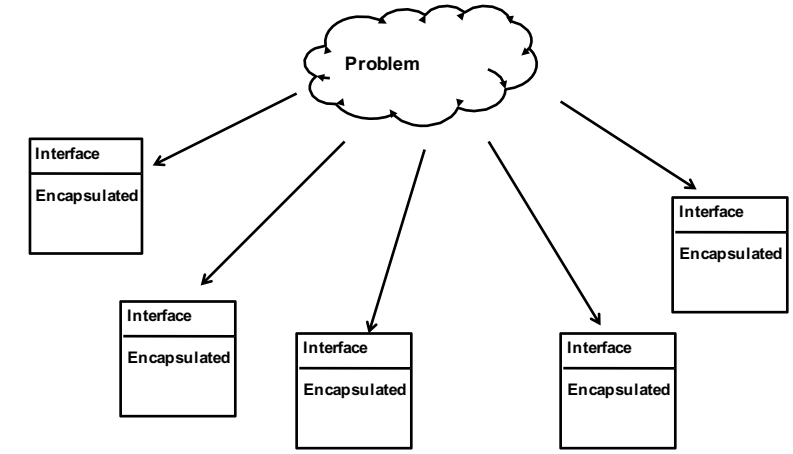
Modularization

- For any large, complex system, must divide the coding into work assignments (WBS)
- Each work assignment is called a “module”
- Properties of a “good” module structure
  - Parts can be designed independently
  - Parts can be tested independently
  - Parts can be changed independently
  - Integration goes smoothly

Modularization Goals

- Reduces complexity, improves manageability
- Coding
  - Can write modules with little knowledge of other modules
  - Replace modules without reassembling the whole system
- Managerial
  - Allows concurrent development
  - Avoids “Mythical Man Month” effect (“adding people to a late software project makes it later”)
- Flexibility/Maintainability
  - Anticipated changes affect only a small number of modules
  - Can calculate the impact and cost of change
- Review/communicate
  - Can understand or review the system one module at a time

Notional Modules

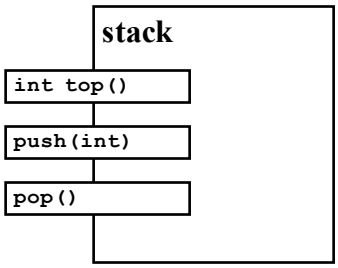


What is a module?

- Concept due to David Parnas (conceptual basis for objects)
- A module is characterized by two things:
  - Its interface: services that the module provides to other parts of the systems
  - Its secrets: what the module hides (encapsulates). Design/implementation decisions that other parts of the system *should not depend on*
- Modules are abstract, design-time entities
  - Modules are "black boxes" – specifies the visible properties but not the implementation
  - May, or may not, directly correspond to programming components like classes/objects
    - E.g., one module may be implemented by several objects

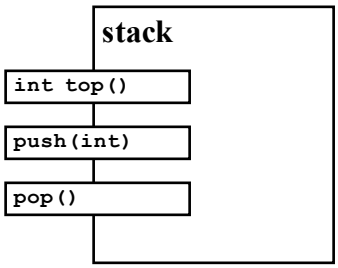
A Simple Module

- A simple integer stack
  - *push*: push integer on stack top
  - *pop*: remove top element
  - *top*: get value of top element
- What information is on the interface?
- What are the secrets?
- What information is missing?
- Why is this an abstraction?



A Simple Module

- A simple integer stack
- The *interface* specifies what a programmer needs to know to use the stack correctly, e.g.
  - *push*: push integer on stack top
  - *pop*: remove top element
  - *top*: get value of top element
- The *secrets* (encapsulated) any details that might change from one implementation to another
  - Data structures, algorithms
  - Details of class/object structure
- A module spec is *abstract*: describes the services provided but allows many possible implementations
- Note: a real spec needs much more than this (discuss later)



Why these properties?

Module Implementer

- The specification tells me exactly what capabilities my module must provide to users
- I am free to implement it any way I want to
- I am free to change the implementation if needed as long as I don't change the interface

Module User

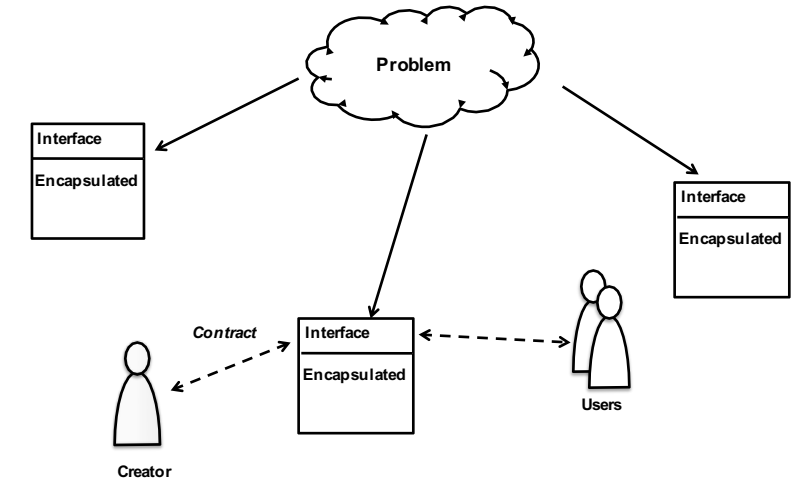
- The specification tells me how to use the module's services correctly
- I do not need to know anything about the implementation details to write my code
- If the implementation changes, my code stays the same

***Key idea*: the abstract interface specification *defines a contract* between a module's developer and its users that allows each to proceed independently**

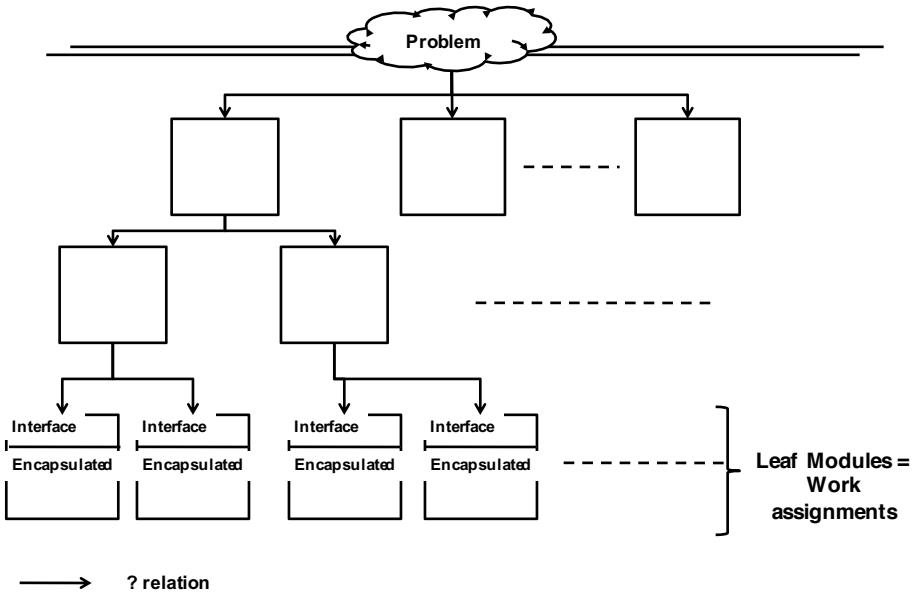
Is a module a class/object?

- The programming language concepts of classes and objects are based on Parnas' concept of modules
- To separate design-time concerns from coding issues, however, *they are not the same thing*
  - A module must be a work assignment at design time, does not dictate run-time structures
  - Coder free to implement with a different class structure as long as the interface capabilities are provided
  - Coder free to make changes as long as the interface does not change
- In simple cases, we will often implement each module as a class/object

Notional Modules



Module Hierarchy



Decomposition Strategies Differ

- How do we develop this structure so that the leaf modules make independent work assignments?
- Many ways to decompose hierarchically
  - Functional: each module is a function
  - Pipes and Filters: each module is a step in a chain of processing
  - Transactional: data transforming components
  - OOD: use case driven development
- Different approaches result in different kinds of dependencies

Use Case Driven OO Process

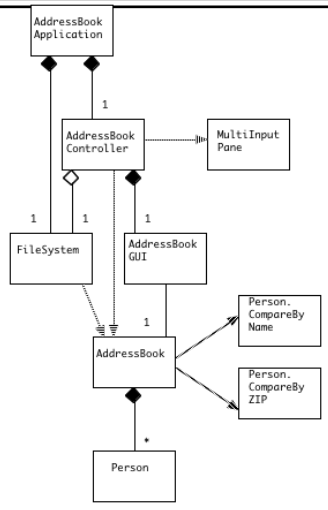
- Address book design: in-class exercise
- Requirements
- Problem Analysis
  - Identify use cases from requirements
  - Identify domain classes operationalizing use cases (apply heuristics)
- OO Design (refinement)
  - Allocate responsibilities among classes
    - CRC Cards (Class-Responsibility-Collaboration)
  - Identify object interactions supporting use cases
    - Sequence or Interaction Diagram for each scenario
  - Identify supporting classes (& associations)
    - Design Class Diagram, relations
- Detailed Design
  - Design class interfaces (class attributes and services)

Decomposition Heuristics

- Heuristics: suppose we create objects by …
  - Underline the nouns
  - Identify causal agents
  - Identify coherent services
  - Identify real-world items
  - Identify physical devices
  - Identify essential abstractions
  - Identify transactions
  - Identify persistent information
  - Identify visual elements
  - Identify control elements
  - Execute scenarios

Use Case Driven OO Process

- Address book design: in-class exercise
- Requirements
- Problem Analysis
  - Identify use cases from requirements
  - Identify domain classes operationalizing use cases (apply heuristics)
- OO Design (refinement)
  - Allocate responsibilities among classes
  - Identify object interactions supporting use cases
  - Identify supporting classes (& associations)
- Detailed Design
  - Design class interfaces (class attributes and services)



Address Book Design Exercise

- Is this a good design?
  - Walk through the handout to understand how the design is derived
    - Understand how use-case-driven OO design works
  - Walk through the design's class diagram and UML class specifications to understand the structure and function of the design
  - Discuss the good and bad points of the design to arrive a team judgment
  - Justify your answer: what is good about it (or bad) and why? What is the role of the MVC pattern?

Lessons

- Without quality requirements there is no basis for choosing between designs
  - i.e., we have no measure for "good"

General OO Objectives

- Manage complexity
- Improve maintainability
- Improve stakeholder communication
- Improve productivity
- Improve reuse
- Provide unified development model (requirements to code)

General OO Principles

- Principles provided to support goals
- Abstraction and Problem modeling
  - Development in terms of problem domain
  - Supports communication, productivity
- Generalization/Specialization (type of abstraction)
  - Inheritance of shared attributes & Delayed Binding (polymorphism)
  - Support for reuse, productivity
- Modularization and Information Hiding
  - Supports maintainability, reuse
- Independence (abstract interfaces + IH)
  - Classes designed as independent entities
  - Supports readability, reuse, maintainability
- Common underlying model
  - OO model for analysis, design, and programming
  - Supports unified development

Additional Design Goals

- Be easy to make the following kinds of change
  - Add additional fields to the entries: for example, fields for someone's email, mobile phone, and business phone
  - Ability to edit the name fields at any time while keeping the associated data
  - As the number of entries gets larger, we will want to be able to search the address book
- Support subsets and extensions
  - Produce a simpler version of the address book with only names and phone #
  - Allow user to keep multiple address books of different kinds (i.e., different fields)
  - Allow the user-defined fields
- Given these explicit and implicit goals, is it a good design?

Exercise: Address Book OOD

- See the class handout
- Use our general OO objectives (implicit) and additional design goals
- Is this a good design with respect to those goals?
  - What is good (or bad) about it?

Questions?