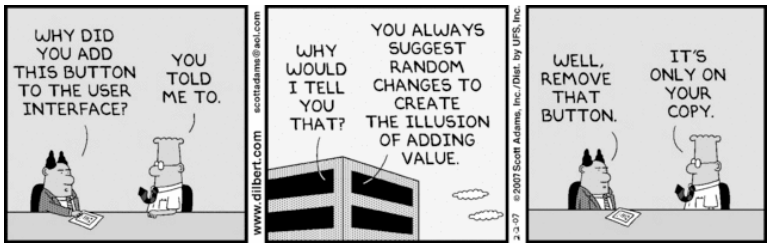
Designing the Module Structure

Standup reports
How do we design to arrive at the desired qualities?
Address Book exercise

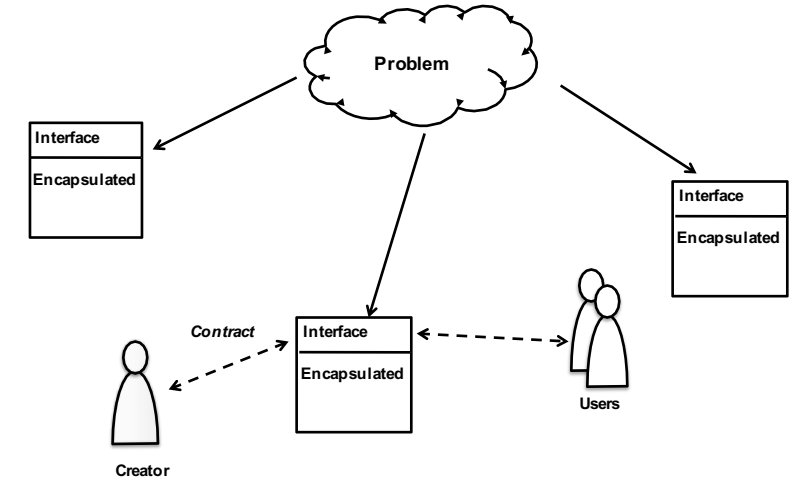


Architecture Design Process

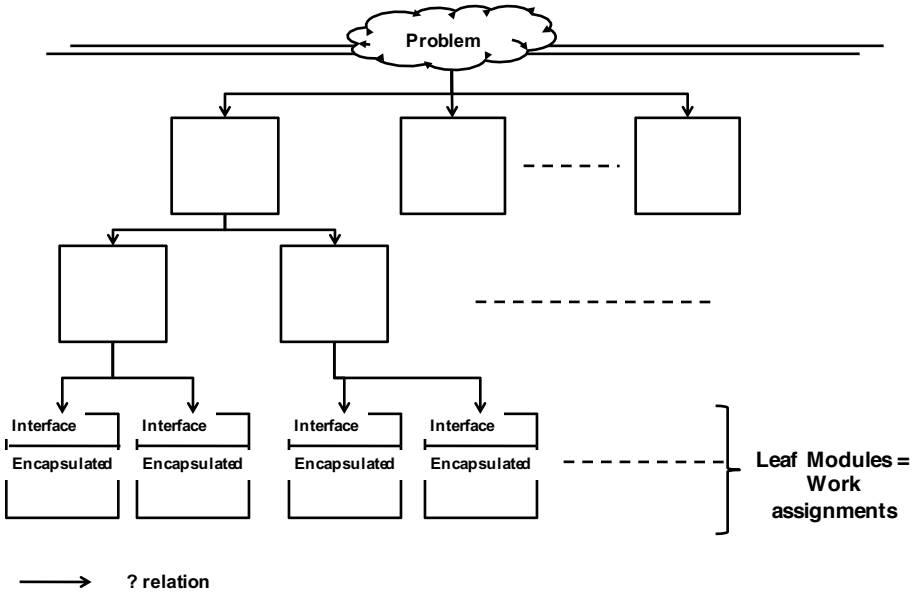
Building architecture to address business goals:

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?)

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

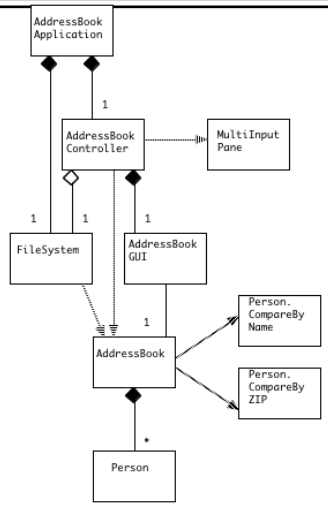
- Address book design: in-class exercise
- Requirements
- Problem Analysis
  - Identify use cases from requirements
  - Identify domain classes operationalizing use cases (apply heuristics)
- OO Design (refinement)
  - Allocate responsibilities among classes
    - CRC Cards (Class-Responsibility-Collaboration)
  - 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?