Software Engineering I CSC-382

Lecture 14

Software Engineering I CS-382

- Chapter 9
- What we will cover: (Design Engineering)
  - Chapter 9 in Pressman
  - We finished learning all the tools for analysis, now we need to make the magic transition to design
  - Goal of chapter is to understand the issues and key concepts of the design phase

Recall the Software Engineering Practices

- Recall the generic process framework elements:

Communication
-Project initiation
-Requirements gathering

Planning
-Estimating
-Scheduling
-Tracking

Modeling
-Analysis
-Design

Construction
-Coding
-Testing

Deployment
-Delivery
-Support

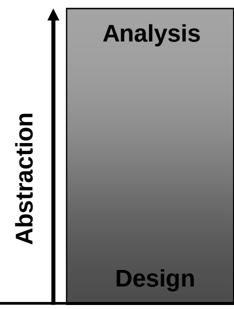
- Here, we'll identify
  - Underlying principles for each
  - A typical task set for each element

Recall the Principles of Analysis Modeling

1) The information domain must be represented and understood.
2) The functions that the software performs must be defined.
3) The behavior of the software as a result of external events must be represented.
4) The models that define #1-#3 must be partitioned to uncover detail in a hierarchical manner.
5) The analysis task moves from essence toward implementation.

Recall Design Modeling

- ***Design models*** represent characteristics of the software that help practitioners to construct it effectively
  - It refines the models begun in the analysis to the point where the software can be constructed



- Elements of the design model
  - Data/Class design
  - Architectural design
  - Interface design
  - Component design

Recall the Principles of Design Modeling

1) Design must be traceable to the analysis model
2) Always consider architecture
3) Design of data as critical as design of functionality
4) Interfaces (external and internal) must be designed
5) User interface should be designed towards **end-user**

Recall the Principles of Design Modeling II

6) Components should exhibit functional independence

7) Components should be loosely coupled

8) Design representation should be easily understood

9) The design model should be developed iteratively

Recall: Design Modeling – Generic Task Set

1. Design appropriate data structures
2. Select an architectural style
3. Partition the analysis model into subsystems and allocate them across this architecture
4. Create a set of design classes or components
5. Design any external interfaces
6. Design the user interface
7. Conduct component-level design
8. Develop a deployment model

Goals to Date for Analysis

- For Structured Analysis:
  - All external sources/sinks of data have been defined
  - All transformations required to map inputs to outputs defined
  - All internal data flows have been defined
  - Transformations have been analyzed to sufficient depth to fully understand system
- For OO Analysis:
  - Basic user requirements have been defined
  - Classes have been identified
  - A class hierarchy has been defined
  - Object-to-object relationships have been analyzed
  - Object behavior has been modeled

What is Software Design ?

- Design is the bridge between the requirements specification (analysis) and the end product.
  - We have viewed our software system as a collection of components with well defined interfaces and responsibilities (or functions)
- The design process usually is defined in 2 steps:
  1. “Top-level design” – this is also called Architectural Design
     - Top level design addresses defining these components and their interactions
  2. “Detailed design”
     - Detailed design addresses the internal design of each of these components

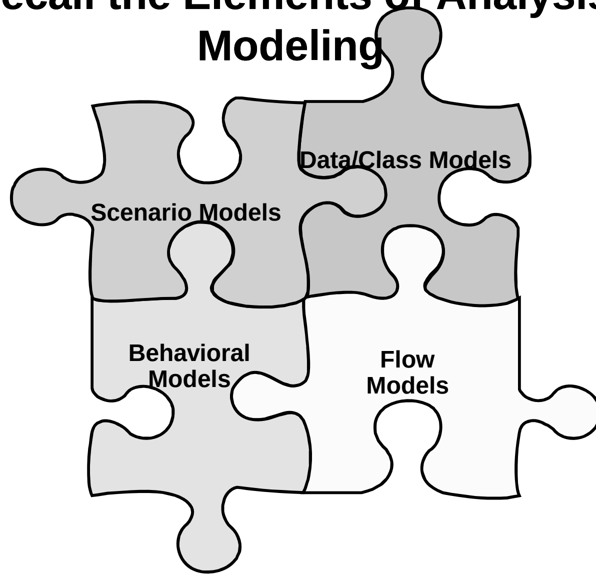
Tools for Our Two Approaches

- For Structured Design we will use:
  - Data and Control Flow Diagrams, State Transition Diagrams, PSPECS and CSPECS (contain our pseudo code), and Structure Charts.
  - Many people also use specification languages we will show a simple one
- For Object Oriented Design we will use:
  - Class diagrams, state diagrams, sequence charts, component diagrams and deployment diagrams
  - Many people also use specification languages here called Object Constraint Languages

Tools for Our Two Approaches II

- Similarity between tools for Analysis and Design but two key differences as well:
  1. In Analysis we created models of the problem, while in Design we are extending these to become models of the solution
  2. In Design the system depends on the model, while in Analysis the model depends on the system.

Recall the Elements of Analysis Modeling

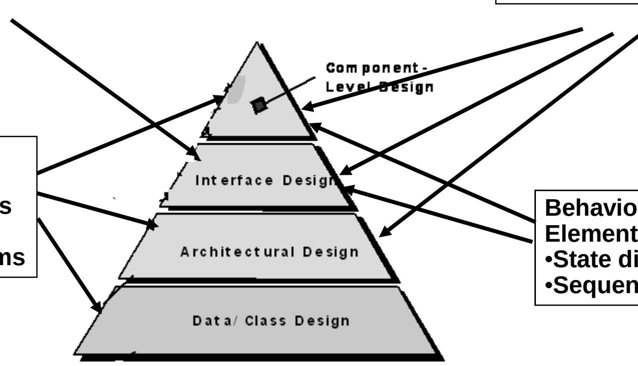


Mapping the Analysis Model to the Design Model

Scenario-based Elements:
- **Use Cases**

Flow-based Elements:
- **Data Flow diags**
- **PSPECS**

Class-based Elements:
- **Class Diagrams**
- **CRC Models**
- **Collab. diagrams**



Behavior-based Elements:
- **State diags**
- **Sequence diags**

Design and Quality

- For a quality product, we must have a quality design:
- The design must implement **all** of the **explicit requirements** contained in the analysis model, and it must accommodate all of the **implicit** requirements desired by the customer.
- The design must be a readable, **understandable** guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a **complete picture** of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

The FURPS Quality Attributes

- **Functionality** – design meets all of the functional requirements
- **Usability** – verified by analyzing aesthetics, consistency, documentation, etc.
- **Reliability** – measure frequency and severity of failures
- **Performance** – verify processing speed and response time, etc.
- **Supportability** – ability to extend adapt and service the final product

Guidelines for a Quality Design I

1. A design should exhibit an architecture that
   - Has been created using recognizable architectural styles or patterns
   - Is composed of components that exhibit good design characteristics
   - Can be implemented in an evolutionary fashion
2. A design should be modular
   - The software should be logically partitioned into elements or subsystems
3. A design should contain distinct representations of data, architecture, interfaces, and components.

Guidelines for a Quality Design II

4. A design use appropriate data structures
   - Suitable to model the classes to be implemented
   - Drawn from recognizable data patterns.
5. A design should have functionally independent components.
6. A design's interfaces should minimize complexity of connections
   - Both between components and with the external environment.

Guidelines for a Quality Design III

7. A design should be derived using a repeatable method
   - Use the information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.
   - Use the templates we have defined to date.

Fundamental Concepts to be Employed in Design

- Abstraction —data, procedure, control
- Architecture —the overall structure of the software
- Patterns —”conveys the essence” of a proven design solution
- Modularity —compartmentalization of data and function

Fundamental Concepts to be Employed in Design II

- Hiding —controlled interfaces
- Functional independence —single-minded function and low coupling
- Refinement —elaboration of detail for all abstractions
- Refactoring —a reorganization technique that simplifies the design

Fundamental Design Concepts - Abstraction

- Abstraction is the key technique for:
  - Managing complexity when we view our design
  - Developing a system architecture and defining the partitioning of the system into modules across that architecture
- Recall we also used abstraction in the analysis phase
  - There we were controlling the complexity of the existing problem so it is easier to discuss,
  - During design, the components do not exist, so we use abstraction to delay defining details of the design

Fundamental Design Concepts – Abstraction II

- Two types of abstraction: Data and Functional
  - E.g. a Vehicle abstracts whether we are discussing cars, trucks, motorcycles,
  - The abstraction of Steering, allows us to plan for the functionality, but to delay its definition, e.g. cars steer very differently from motorcycles
  - Recall in 175/275 you used function stubs to hold off on design detail

Fundamental Design Concepts– Architecture & Patterns

- Architecture defines the overall structure of the software
  - It must be consciously defined and not accidentally derived
  - It defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.
  - It provides a good high-level ‘abstract’ view of the system to help guide the remaining design decisions
  - It is a good practice to draw upon repeatable patterns that are commonly encountered in the design of families of similar systems.

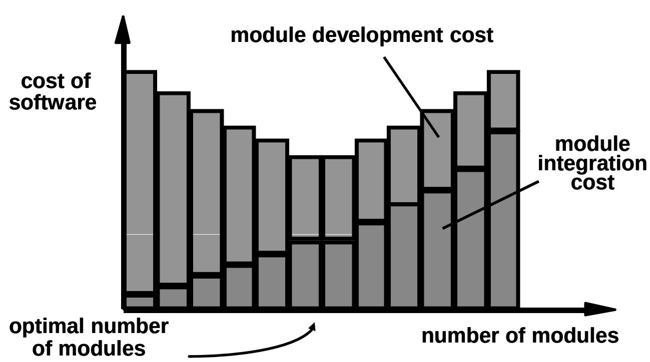
Fundamental Design Concepts - Modularity

- A design is modular if it is composed of discrete components
  - Each component can then be implemented separately
  - Each component can be well-understood

- Software cannot be made modular simply by breaking it into pieces however!
  - Each module must be a well-defined abstraction
  - Each module must have well defined interfaces
  - There is an optimal point for modularization

Modularity: Trade-offs

What is the "right" number of modules for a specific software design?



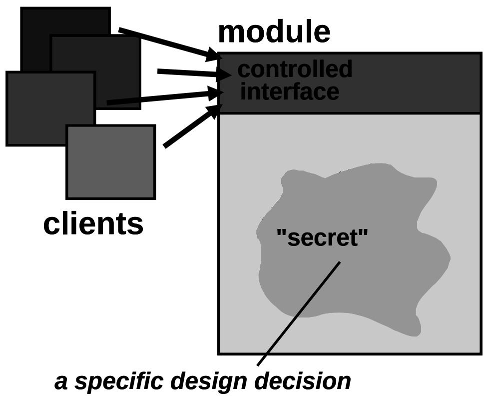
Fundamental Design Concepts - Information Hiding

- Information hiding is like the implementation side of abstraction
  - Abstraction allows us to define the elements of the software
  - Information hiding then enforces how much detail any other element of the system will know about the other elements
  - It reduces the likelihood of “side effects”
  - Limits the global impact of local design decisions

Fundamental Design Concepts - Information Hiding II

- Info hiding is critical for effective modularity
  - Leads to encapsulation—an attribute of high quality design
  - Emphasizes communication through controlled interfaces
  - If we have a modular design but then allow all the elements to communicate at any level desired then we still have a very complex design

Fundamental Design Concepts - Information Hiding



Fundamental Design Concepts - Functional Independence

- Functional independence is strongly related to modularity and information hiding, and abstraction
- Want to design modules with:
  - ‘single minded’ focus
  - ‘aversion’ to excessive interaction (asocial!)
- There are two aspects to Functional Independence
  1. Cohesion – defines how closely related the set of functions are that a module must perform
  2. Coupling – defines how much interaction with other modules is required for a module to perform its desired role.

Functional Independence – Levels of Cohesion

Increasing cohesion

- Coincidental – no meaningful relationship between the functions within the module
- Logical – some logical relationship exists between the functions, e.g. all inputs or all outputs, etc.
- Temporal – elements execute together, e.g. initialization, termination, etc.
- Procedural – all the elements belong to a common procedural element, e.g. all reside in the same while loop

Jalote

Functional Independence – Levels of Cohesion II

Increasing cohesion

- Communicational – all the elements refer to the same inputs or outputs, e.g. format and print report, etc.
- Sequential – outputs from one element are the inputs for the next element in the module, e.g. group data flow diagram elements together
- Functional – all the elements in the module work to perform a single function

Jalote

Functional Independence – Levels of Coupling

- Three factors influence the coupling complexity
  1. Interface complexity – e.g. number of parameters passed to a function
  2. Type of connection – e.g. interface through one well defined entry, or can every module interface differently with it
  3. Type of communication – is it data, control, or both (data is simplest)

Functional Independence – Levels of Coupling II

Degree of Coupling				
	Low			
	High			

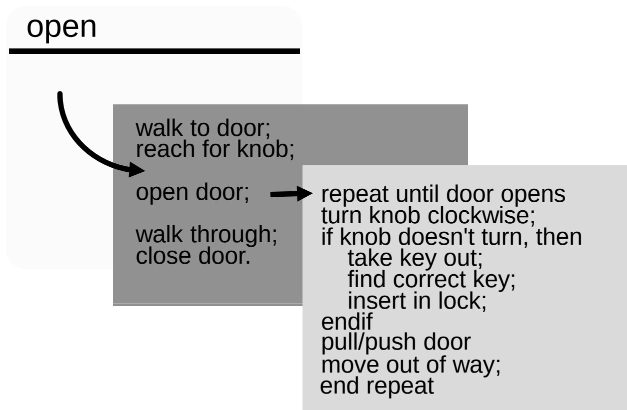
Fundamental Design Concepts - Stepwise Refinement

- Step-wise refinement is the basis of Top-Down design
  - Top-down defines the system at its highest level of abstraction and then continues to decompose the system into components
- Recall, Bottom-up design is another approach –
  - It starts with the most primitive elements and then builds the system up from these building blocks
  - When building from an existing system this is often better

Fundamental Design Concepts - Stepwise Refinement II

- In reality, these two are combined
  - Recall, throughout analysis I mentioned we work our way down, and then go back up when we see something missing or inconsistent
  - Usually we first establish an abstraction layer for our application, and then we perform a mix of top-down and bottom up from there

Fundamental Design Concepts - Stepwise Refinement



Fundamental Design Concepts – Refactoring

- Fowler [FOW99] defines refactoring in the following manner:
  - "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its **internal structure**.”

Fundamental Design Concepts – Refactoring II

- When software is refactored, the existing design is examined for:
  - Redundancy
  - Unused design elements
  - Inefficient or unnecessary algorithms
  - Poorly constructed or inappropriate data structures
  - Any other design flaw you can correct to yield a better design.
- Refactoring is key for our iterative up and down approach
  - You never get it right the first time, so plan for a refactoring step

Fundamental Design Concepts – Transitioning from Analysis to Design

- To date we have developed a set of models during our analysis modeling – people call these analysis models
- In the design phase we will develop Design Models
  - We just need to **refine** our existing models not re-do them
    - What we will need is to add classes (OO) or other transforms (SAD) to handle the implementation type issues (**Infrastructure & Control**)

For Next Class

- Finish studying Chapter 9