605.601 Foundations of Software Engineering
Fall 2020

Module 05: Software Design Foundation

Dr. Tushar K. Hazra

tkhazra@gmail.com

(443)540-2230

Course Module 05: Software Design Foundation

Topics for Discussion

- Requirements
- Use Cases
- Design – Definition and Major Steps
- Design Principles
- Architectures - Introduction

Course Module 05: Software Design Foundation

Recapitulation

- Types of Requirements
  - A functional requirement describes the required behavior in terms of required activities and the state of each entity before and after an activity
  - A quality requirement (or nonfunctional requirement) is a characteristic that the system must possess
    - Fit criteria quantify the extent to which each requirement must be met

Course Module 05: Software Design Foundation: Recap

- Prototyping
  - Rapid prototyping builds software to answer questions about the requirements
    - Partial solution that aids in understanding the requirements or evaluating design alternatives
  - Throwaway Prototype
    - A throwaway prototype is developed to learn about the problem
      - Never intended to be part of the delivered software
  - Evolutionary Prototype
    - An evolutionary prototype is developed to answer questions and to incorporate into the final product
      - Requires more care because quality requirements must eventually be met

Course Module 05: Software Design Foundation: Recap

- Requirements Engineering
  - Inception: Establish the need for a software system Elicitation Collect requirements from the customer Analysis Model the desired behavior of the system
  - Negotiation: Agree on a deliverable system that is realistic for developers and customers
  - Specification: Document the behavior of the proposed system Validation Check that the specification matches the customer's requirements (i.e., expectations)
  - Requirements management: Develop a way to manage and maintain the terms of negotiation – be reasonable

Course Module 05: Software Design Foundation : Recap

- Validating Requirements
  - Is each requirement consistent with the overall objective for the system/product?
  - Is each requirement bounded and unambiguous?
  - Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
  - Is each requirement testable, once implemented?
  - Has the requirements model been "partitioned" in a way that exposes progressively more detailed information about the system?

Course Module 05: Software Design Foundation

- Use Cases
  - A collection of user scenarios that describe the thread of usage of a system
  - Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
  - Each scenario answers the following questions:
    - Who is the primary actor, the secondary actor(s)?
    - What are the actor's goals?
    - What preconditions should exist before the story begins?
    - What main tasks or functions are performed by the actor?
    - What extensions might be considered as the story is described?
    - What variations in the actor's interaction are possible?
    - What system information will the actor acquire, produce, or change?
    - Will the actor have to inform the system about changes in the external environment?
    - What information does the actor desire from the system?
    - Does the actor wish to be informed about unexpected changes?

Course Module 05: Software Design Foundation

- What is Design?
  - Design is the principles, concepts, and practices that lead to the development of a high-quality system

Questions about whether design is necessary or affordable are quite beside the point: design is inevitable. The alternative to good design is bad design, not no design at all. (Douglas Martin)

Course Module 05: Software Design Foundation

- Major Steps in Design
  - Moving from a high-level view of system to implementation-level details. . .
    1. Representation of system architecture
    2. Modeling of interfaces (with users, other systems, and internal components)
    3. Design of individual components
- Result is an early representation of software that can be assessed for quality

Course Module 05: Software Design Foundation

- Principles of Design
  - Mitch Kapor, the creator of Lotus 1-2-3, presented the following "software design manifesto" in Dr. Dobbs Journal:

    Good software design should exhibit
  - Firmness A program should not have any bugs that inhibit its function
  - Commodity A program should be suitable for the purposes for which it was intended
  - Delight The experience of using the program should be pleasurable one

Course Module 05: Software Design Foundation

- Design and Quality
  - 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.

Course Module 05: Software Design Foundation

- Quality Guidelines I
  - 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, and
    - ... can be implemented in an evolutionary fashion
      - ... For smaller systems, design can sometimes be developed linearly
  - A design should be modular: the software should be logically partitioned into elements or subsystems
  - A design should contain distinct representations of data, architecture, interfaces, and components.

Course Module 05: Software Design Foundation

- Quality Guidelines II
  - A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns
  - A design should lead to components that exhibit independent functional characteristics
  - A design should lead to interfaces that reduce the complexity of connections between components and with the external environment
  - A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis
  - A design should be represented using a notation that effectively communicates its meaning

Course Module 05: Software Design Foundation

- Design Principles
  - The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.
  - The design should exhibit uniformity and integration.
  - The design should be structured to accommodate change.
  - The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
  - Design is not coding, coding is not design.
  - The design should be assessed for quality as it is being created, not after the fact.
  - The design should be reviewed to minimize conceptual (semantic) errors.

Course Module 05: Software Design Foundation

- Fundamental Concepts
  - Abstraction data, procedure, control Architecture the overall structure of the software
  - Patterns "conveys the essence" of a proven design solution Separation of Concerns any complex problem can be more easily handled if it is subdivided into pieces
  - Modularity compartmentalization of data and function Information Hiding controlled interfaces
  - Functional Independence single-minded function and low coupling Refinement elaboration of detail for all abstractions
  - Aspects a mechanism for understanding how global requirements affect design
  - Refactoring a reorganization technique that simplifies the design
  - Design Classes provide design detail that will enable analysis classes to be implemented

Course Module 05: Software Design Foundation

- Data Extraction

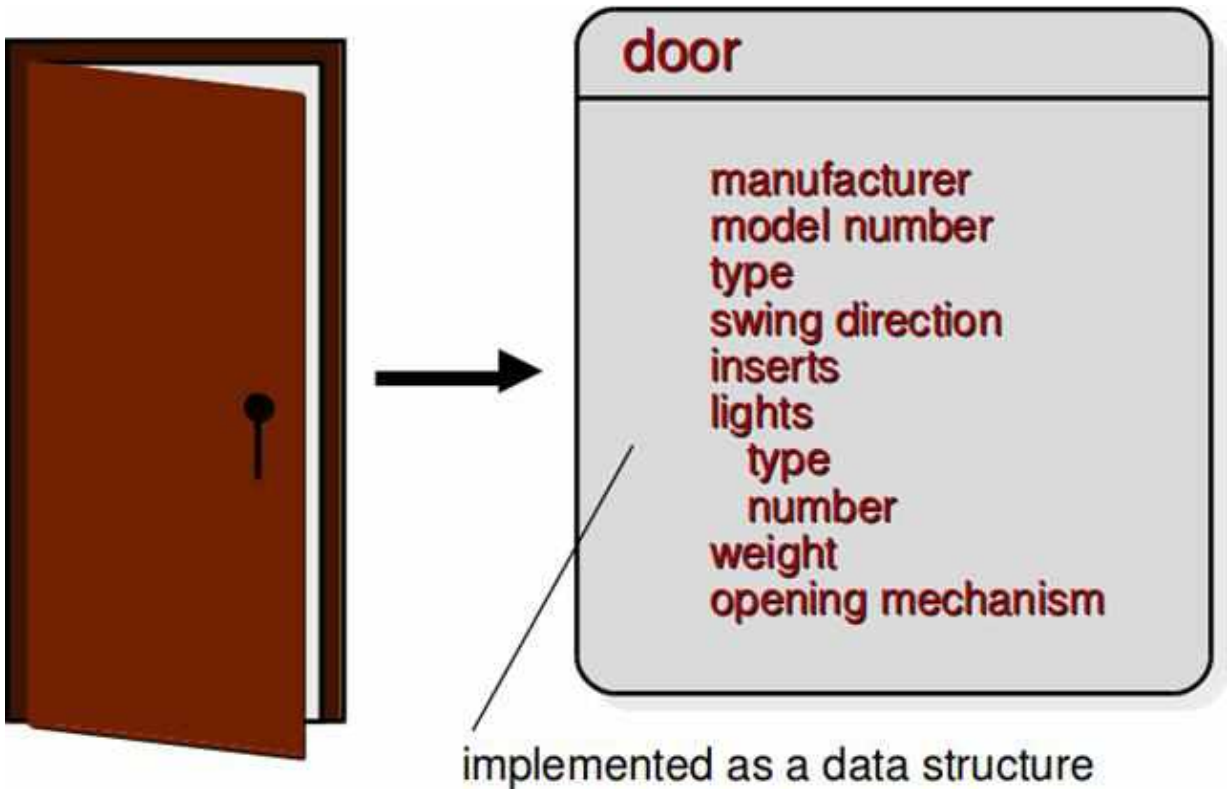


Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Course Module 05: Software Design Foundation

- Procedural Abstraction

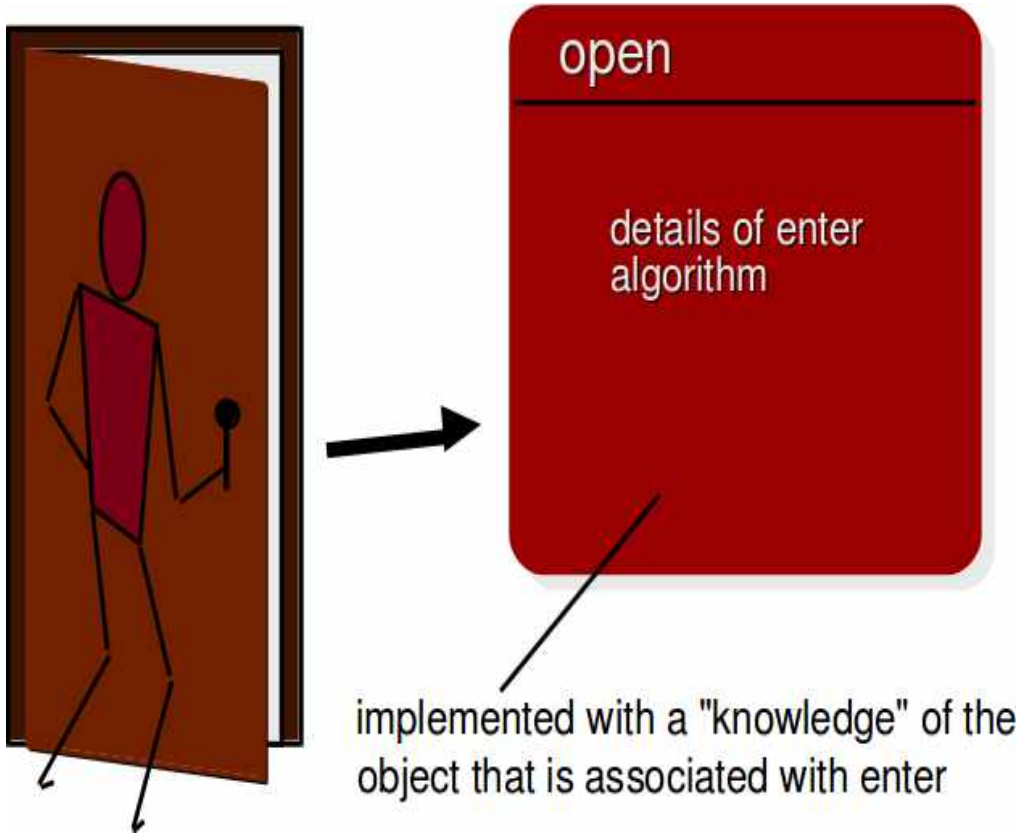


Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Course Module 05: Software Design Foundation

- Architecture
  - "The overall structure of the software and the ways in which that structure provides conceptual integrity for a system." (Shaw and Garlan, 1995)
  - Structural properties This aspect of the architectural design represents the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.
  - Quality properties The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.
  - Families of related systems The design should have the ability to reuse architectural building blocks.

Course Module 05: Software Design Foundation

- Patterns
  - Design Pattern Template
    - Name describes the essence of the pattern in a short but expressive name
  - Intent describes the pattern and what it does Motivation provides an example of the problem
  - Applicability notes specific design situations in which the pattern is applicable
  - Participants describes the responsibilities of the classes that are required to implement the pattern
  - Collaborations describes how the participants collaborate to carry out their responsibilities
  - Consequences describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented

Course Module 05: Software Design Foundation

- Separation of Concerns
  - Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved or optimized independently
  - By separating concerns into smaller, and therefore more manageable pieces, a problem takes less effort and time to solve.

A concern is a feature or behavior that is specified as part of the requirements model for the software

Course Module 05: Software Design Foundation

- Modularity
  - Modularity is the single attribute of software that allows a program to be intellectually manageable (Myers, 1978)
  - Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a software engineer.
    - ... The number of control paths, span of reference, number of variables, and overall complexity make understanding close to impossible.
- In almost all instances, you should break the design into many modules, hoping to make understanding easier and as a consequence, reduce the cost required to build the software.

Course Module 05: Software Design Foundation

- Information Hiding

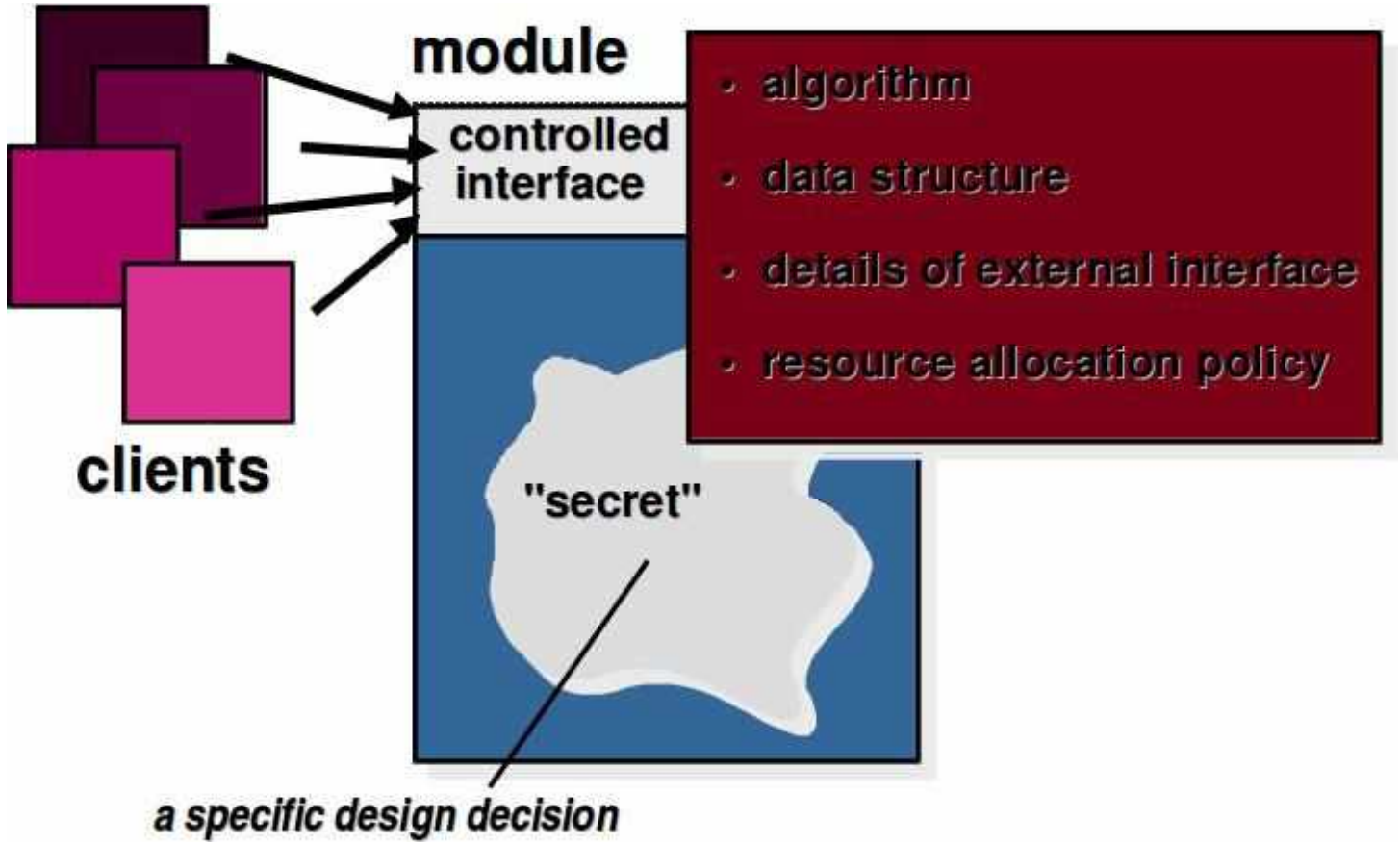


Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Course Module 05: Software Design Foundation

- Why Information Hiding?
  - Reduces the likelihood of "side effects"
  - Limits the global impact of local design decisions
  - Emphasizes communication through controlled interfaces
  - Discourages the use of global data
  - Leads to encapsulation—an attribute of high quality design
  - Results in higher quality software

Course Module 05: Software Design Foundation

- Stepwise Refinement

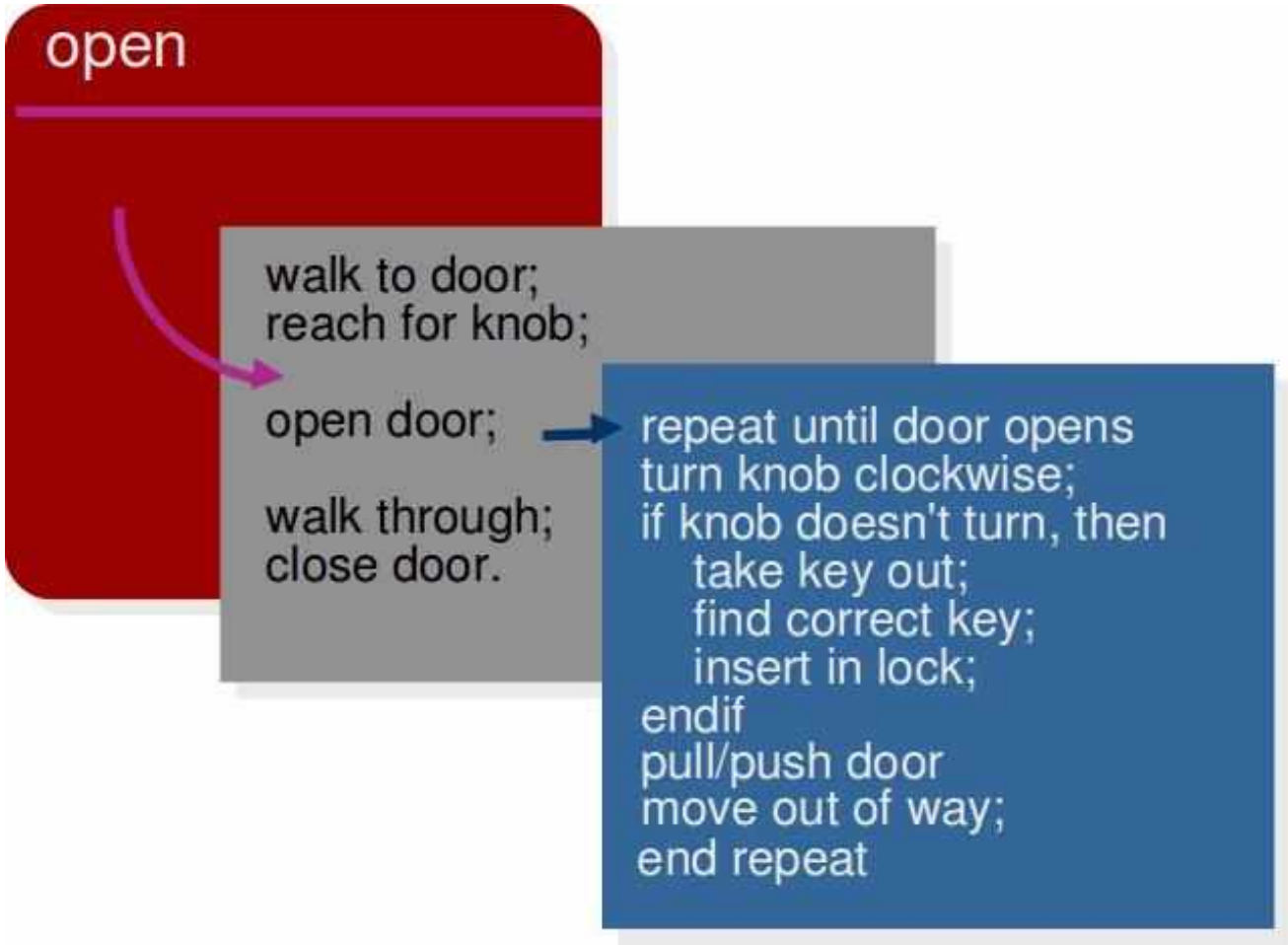


Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Course Module 05: Software Design Foundation

- Functional Independence
  - Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
  - Cohesion An indication of the relative functional strength of a module.
    - A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing.
  - Coupling An indication of the relative interdependence among modules.
    - Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

Course Module 05: Software Design Foundation

- Refactoring
  - Definition (Refactoring)
  - 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. (Fowler, 1999)

- When software is refactored, the existing design is examined for. . .
  - redundancy,
  - unused design elements,
  - inefficient or unnecessary algorithms,
  - poorly constructed or inappropriate data structures, and
  - any other design failure that can be corrected to yield a better design.

Course Module 05: Software Design Foundation

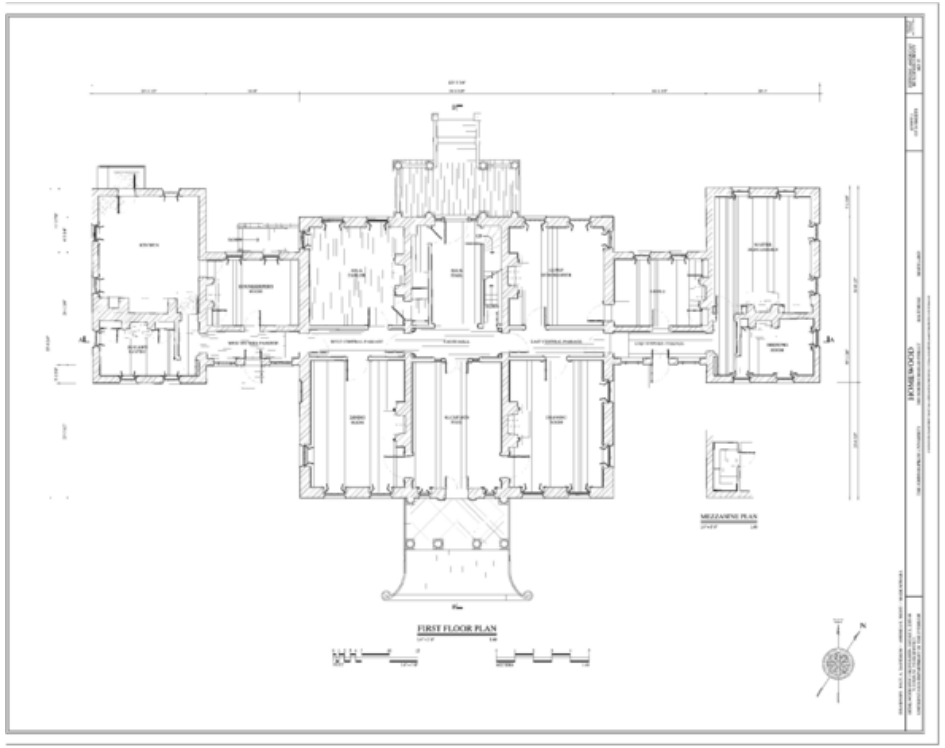
- Object-Oriented Design Concepts
  - Design classes
  - Inheritance All responsibilities of a superclass is immediately inherited by all subclasses
  - Messages Stimulate some behavior to occur in the receiving object
  - Polymorphism A characteristic that greatly reduces the effort required to extend the design

Course Module 05: Software Design Foundation

- Design Classes
  - Analysis classes are refined during design to become entity classes
  - Boundary classes are developed during design to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.
    - .., Boundary classes are designed with the responsibility of managing the way entity objects are represented to users.
  - Controller classes are designed to manage
    - .., the creation or update of entity objects;
    - .., the instantiation of boundary objects as they obtain information from entity objects;
    - .., complex communication between sets of objects; or
    - .., validation of data communicated between objects or between the user and the application.

Course Module 05: Software Design Foundation

- Architectural design is information driven from data, functional, and behavioral domains of requirements analysis
- Blueprint for software



Course Module 05: Software Design Foundation

- Why Architecture?
  - The architecture is not the operational software. Rather, it is a representation that enables a software engineer to
    - analyze the effectiveness of the design in meeting its stated requirements
    - consider architectural alternatives at a stage when making design changes is still relatively easy, and
    - reduce the risks associated with the construction of the software.

Course Module 05: Software Design Foundation

- Why is Architecture Important?
  - Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
  - The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
  - Architecture "constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together" (Bass et al., 2003).