ENPM 809W
Introduction to Secure Software Engineering

Gananand Kini

Lecture 2.5

Software Design

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

Outline

- What is Software Design?
- Elements of software design process
- Resources to learn more

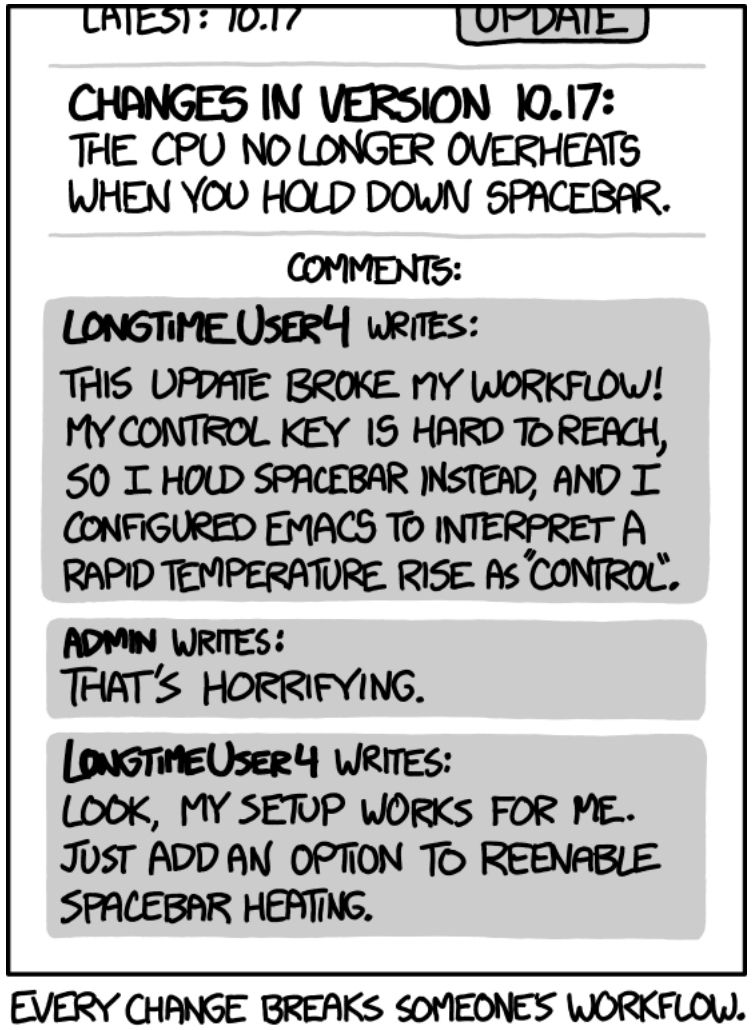
What is Software Design?

Software Design

- A process that transforms requirements into some form that makes it easy to implement as part of a software system.
- An iterative process – you start with an informal design outline, then try an informal design, then you start formalizing some of the design elements and then finally create a finalized design. However, you will always find issues and will then start the process all over again.
- As mentioned before in Week 2, design is a “wicked problem”.

Source: Telkar, N. Software Design Techniques Student Project for CSCI-5828. https://home.cs.colorado.edu/~kena/classes/5828/s10/presentations/softwaredesign.pdf.

Why reiterating is hard in real life …



Sources:
1. XKCD: Workflow. https://xkcd.com/1172/. Retrieved October 4, 2021.

Software System Requirements come first…

- **The library has volunteers, IT staff and patrons that will use the Library Book Lending System (LBLS).**
- **The LBLS has the following use cases:**
  - A volunteer must be able to:
    - Add Books
    - Remove Books for Auction
    - Submit timesheets
    - …
  - A patron must be able to:
    - Check out books
    - Return books
    - Check late penalty balance
    - …
  - The IT staff must be able to:
    - View Logs
    - Manage databases
    - Ability to run scripts to perform maintenance tasks
    - …

Software System Requirements come first…

- **Functional requirements:**
  - Patron's use case of Checkout Books:
    - Requires that a Patron have an account with the library.
    - Requires that the library have enough information to contact the Patron by address, email or Phone
  - Volunteer's use case of Add Books to Library Inventory:
    - Requires the library have a inventory management system
    - Requires the library allow storage of books before they are put into circulation
    - Requires tracking of books using Hewey Decimal system?
    - Requires tracking of books using barcodes?
    - …

Elements of Software Design Process

Caveat Emptor

- This is just a brief summary/intro to the field of software design.
- Focuses only on component based design and not other architectures.
- This is by no means complete and the presenter encourages you to go read good articles/books on the topic specified in the Resources slide.

Software Design

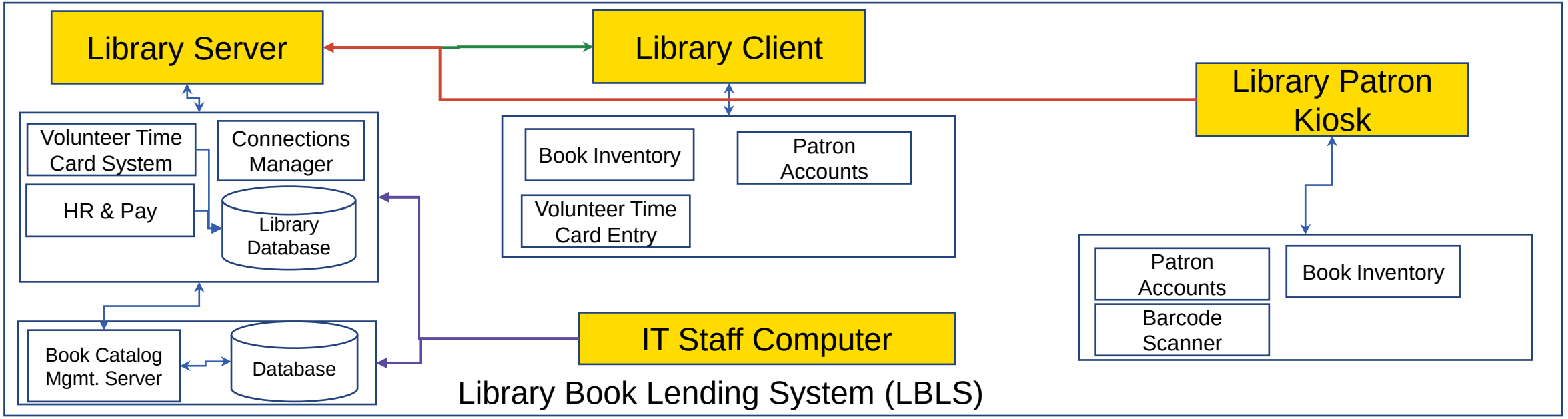
- **Break your requirements and use cases for the software system into sub-systems with service offerings and components. (Architectural design)**

- **Abstraction – Components can be created using conceptual entities that facilitates solving a problem. What are the essential characteristics required for the entity?**

- **Modularization – Decompose the system into *finer grained* components.**

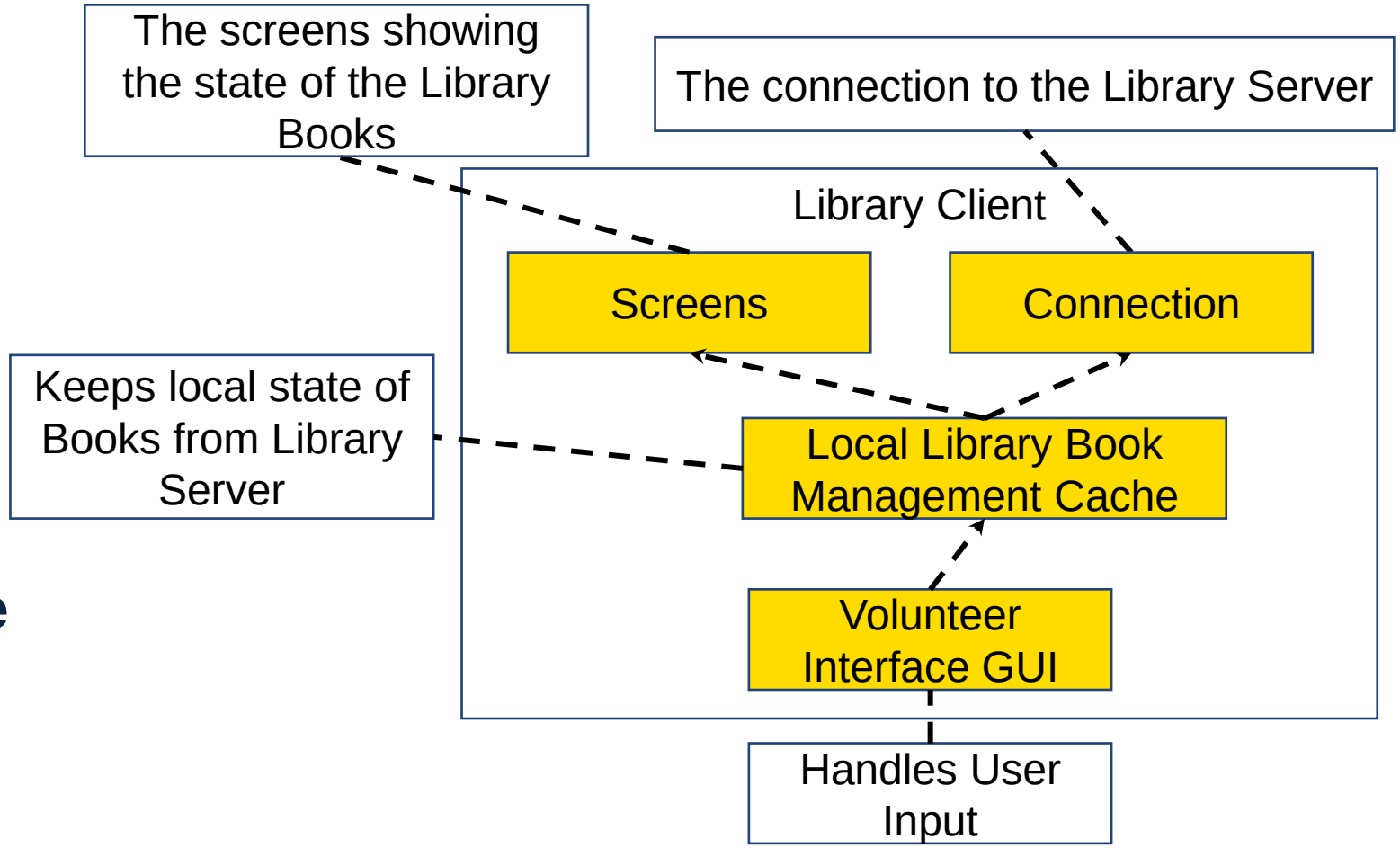
Architectural Design

- **You can take the requirements you have and start outlining a high level architecture that:**
  - Breaks down the software system into its sub-systems and components
  - Identifies the control and communication channels
- **As an example, for the library book system, a possible architecture:**



Architectural Design

- **For each sub-system, an abstract specification of the services it provides and the constraints under which it must operate is produced…**
- **Services include:**
  - Visual representation
  - Communication Layer
  - User I/O
  - Etc.
- **Here Yellow boxes are sub-systems**
- **The other boxes are services provided by the sub-systems.**



Interface Design

- **For each sub-system, its interface is designed and documented.**
- **For example,**
- **The Library Client can:**
  - GetBookInventory()
  - UpdateBookInventory()
  - AddToBookInventory()
  - GetPatronInfo()
  - GetPatronDues()
  - …
- **The Library Patron Kiosk can:**
  - CheckoutBookFromInventory()
  - ReturnBookToInventory()
  - GetLateDues()
  - …

Component Design

- **Services are allocated to different components. For example:**
  - Local state of Books from Library Server in the Library Client might be managed by a BookInventoryManagement component.
- **This phase of design entails detailed implementation design of the interfaces that were identified in the previous interface design.**
  - For example, now the BookInventoryManagement component implements:
    - AddBookToInventory()
    - CheckoutBookFromInventory()
    - ReturnBookToInventory()
    - …
- **Services allocated to each sub-system as they are going to be implemented are also designed in this phase.**

Data Structure Design

- **The data structures that are needed in the system implementation are designed in detail and specified during this phase.**
- **Example:**
- **Book**
  - Title
  - ISBN
  - Authors
  - Publisher
  - …
- **Library**
  - Library Branch Unique Identifier
  - Library Branch Name
  - Library Branch Address
  - …

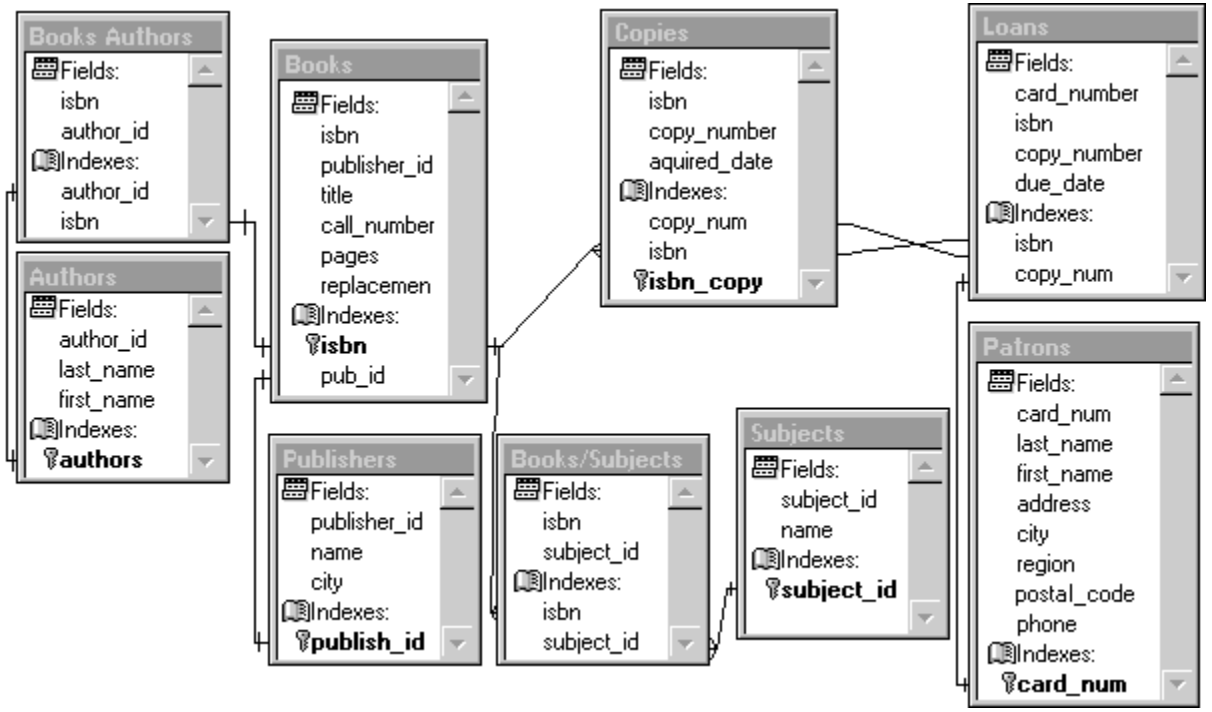


Fig. An example Database Design for managing Library Books

Source: Qureshi, O. Stackoverflow.com. https://stackoverflow.com/questions/10884677/a-library-database.

Algorithm Design

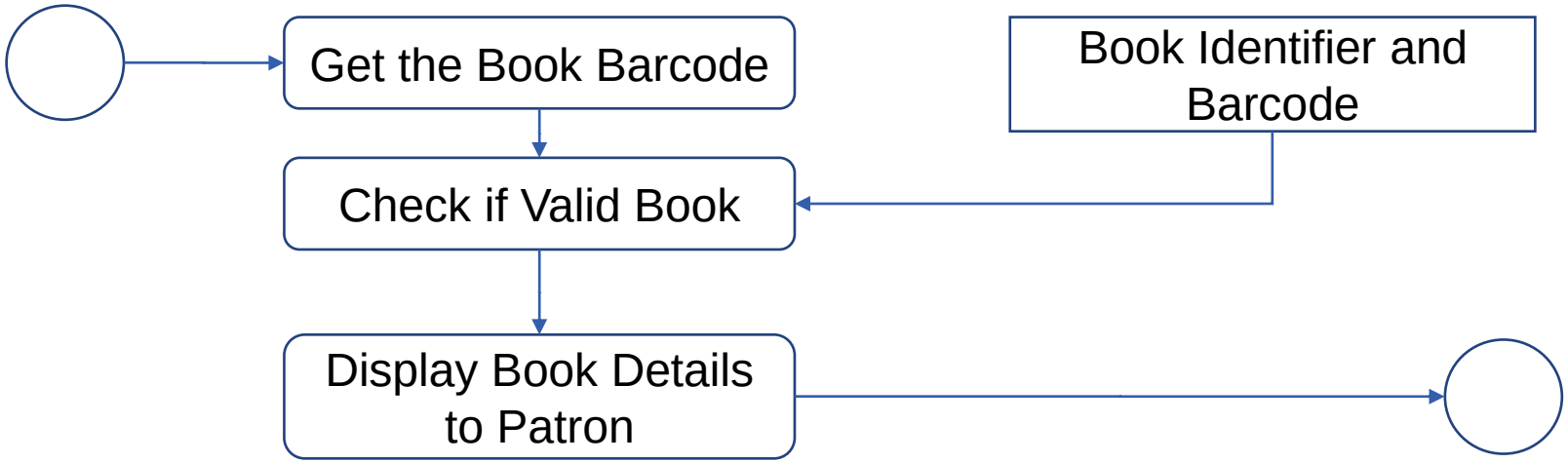
- Here any services that require the use of complex logic are designed.
- This might involve designing how to get the results from the logic processing in polynomial time.

Structured Method of Software Design

- **Sets of Notations and Guidelines for software design.**
- **Supports some or all of the following models for a system:**
  - A data-flow model
  - An Entity-Relationship model
  - A structural model
  - An object-oriented model

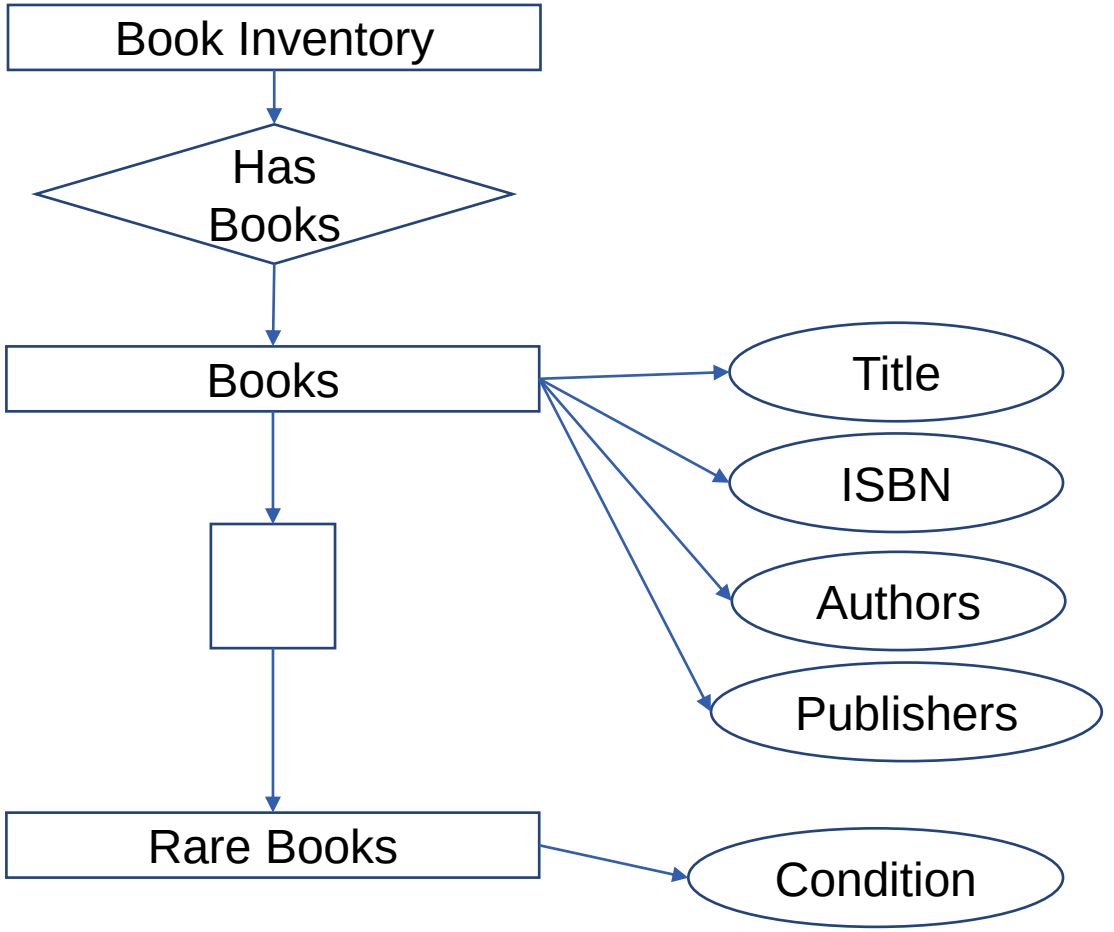
Data Flow model

- Circles represent user interactions with the system which either provide input or receive output.
- Rectangles represent Data Stores.
- Rounded Rectangles represent functions that transform input into output.
- Arrows show the direction of data flow.
- Example:



Entity Relationships Model

- A Rectangle is used to represent an Entity.
- An Oval is used to represent either an Entity or a Relationship Attribute.
- A diamond is used to represent a relationship between Entities.
- A square is used to represent an inheritance relationship where the Derived Entity inherits the attributes of its related Entity.
- Example:



Component Model – Object Oriented View

- When doing component design as mentioned earlier, you could you split your services to be implemented using object oriented components – Classes.
- You implement the service interfaces using methods and attributes as members of the class.
- An example of this is your typical Object-Oriented software with relational databases containing objects to represent your data structures and typical CRUD operations.

Component Model – Functional View

- The design can also make use of a functional view in its component model where the processing logic is broken up into pieces based on the data structures required to implement the logic and an interface that enables the component to be invoked and data passed to it.
- An example of this is machine learning libraries where you provide data to the library and then you are able to invoke functions on it to compute and implement your processing logic.

Component Model – Process-related View

- The design could also be broken up into a process-related view where you have a library of existing components and as you build your software architecture you choose components from that library to populate your architecture.
- An example of this is UI made with building blocks like dialog boxes, modals, buttons etc.

Software Design

- **Now that you have components design and a model, you have to iterate based on whether the software design is “good”.**
- **If it is not, you repeat the steps and process from the beginning while trying to meet the new “good” criteria.**
- **Components in a design can have the following properties:**
  - Reusability – Components are typically designed to be reused in different applications.
  - Replaceable – Components maybe freely substituted with other similar components.
  - Not context specific – Components are designed to operate in different environments and contexts.
  - Extensible – Components can be extended from existing components to provide new behavior.
  - Encapsulated – A Component depicts the interfaces that allow a caller to use its functionality and does not expose internal variables or internal state.
  - Independent – Components are designed to have minimal dependencies on other components.

Source: Tutorialspoint: Component Based Architecture. https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm.

Software Design contd…

- **More component properties:**
  - Cohesive – Components will typically have all related functionality together within itself and rely less on data structures and functionality from other components.
  - Component interaction takes place by method invocation, asynchronous invocation, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.
  - For a server based class or component, specialized interfaces are created that serve major categories of clients. Only those operations that are relevant to a particular category of clients should be in the interface.

Source: Tutorialspoint: Component Based Architecture. https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm.

SOLID Principles

- **S – Single Responsibility Principle: The component only has one responsibility. For example: if the BookCatalogManagement component manages books, it should not include printing Book information.**
- **O – Open/Closed Principle: The component should be open for extension but closed for modification.**
- **L – Liskov Substitution Principle: In the strictest sense here, this principle means that a child or derived component can be replaced in lieu of its parent without causing problems in the software system. The design should ensure this.**
- **I – Interface Segregation Principle: This principle means that separating interfaces to the least set of methods required to implement an interface such that implementors of the interface don't have to implement unnecessary methods.**
- **D – Dependency Inversion Principle: This principle means that components should depend on abstractions and not concretions. A high level module should not depend on a low level module, instead depending on abstractions. Example: IDBProvider in WebGoat which injects the appropriate DB interface at runtime.**

Source: Robert C. Martin. Design Principles and Design Patterns. 2000.

Resources

- **Robert C. Martin – http://cleancoder.com**
  - Paper: https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf

- **Software Engineering Design: Theory and Practice by Carlos Otero. ISBN 9781439851685. June 11, 2012.**

- **IEEE 1016 Working Group: http://www.iso-architecture.org/ieee-p1016/**