Chapter 1 - Software Design Fundamentals

Lecture Outline

- Introduction to Software Design
- Levels of Design
- Software Design Process
- Software Process Models
- Software Design Approaches
- Software Design Concepts/Principles

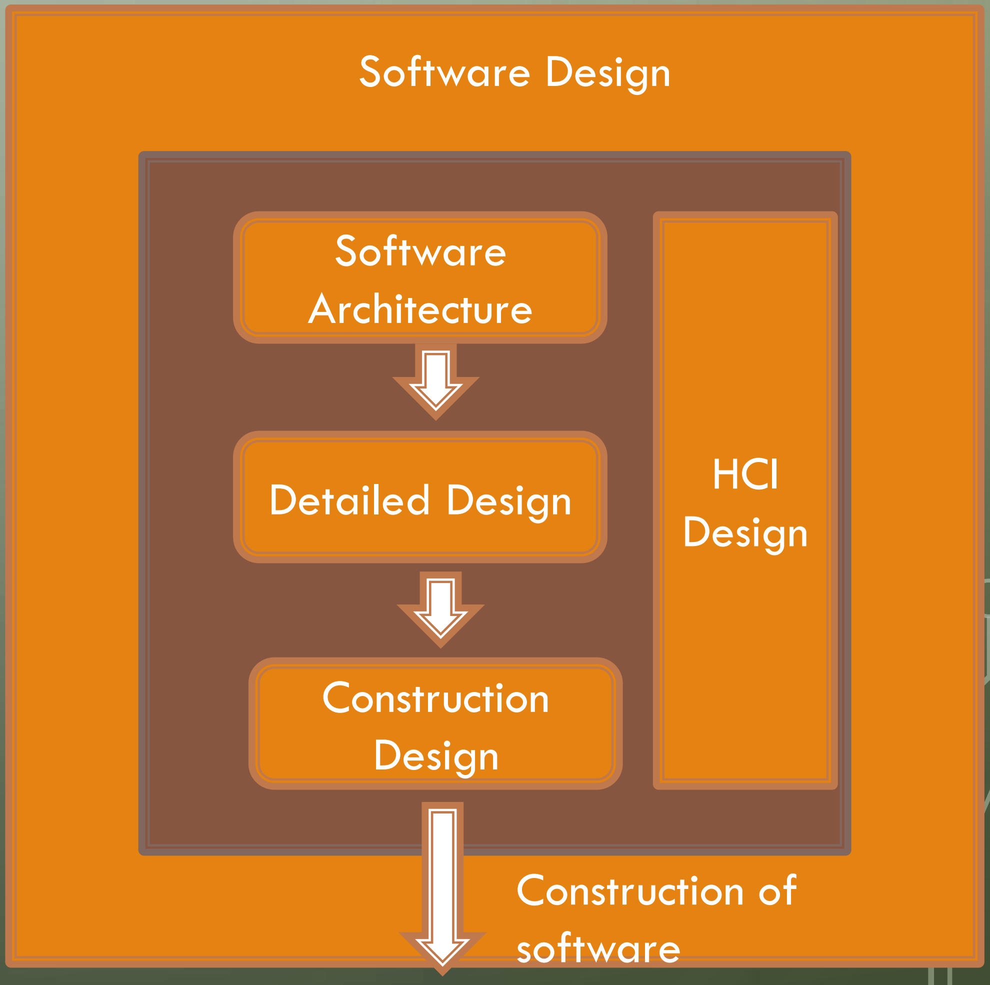
INTRODUCTION TO SOFTWARE DESIGN

- **Requirements specification was about the WHAT the system will do.**
- **Design is about the HOW the system will perform its functions.**
  - Provides the overall decomposition of the system
  - Allows to split the work among a team of developers
  - Also lays down the groundwork for achieving non-functional requirements (performance, maintainability, reusability etc.)
  - Takes target technology into account (e.g. kind of middleware, database design etc.)

LEVELS OF DESIGN

The process of software design can be divided into levels:

- Architectural Design (Software Architecture)
- Detailed Design
- Construction Design



ARCHITECTURAL DESIGN

ARCHITECTURAL DESIGN

- A.k.a **Software Architecture**
- **The term Architecture is defined as :**

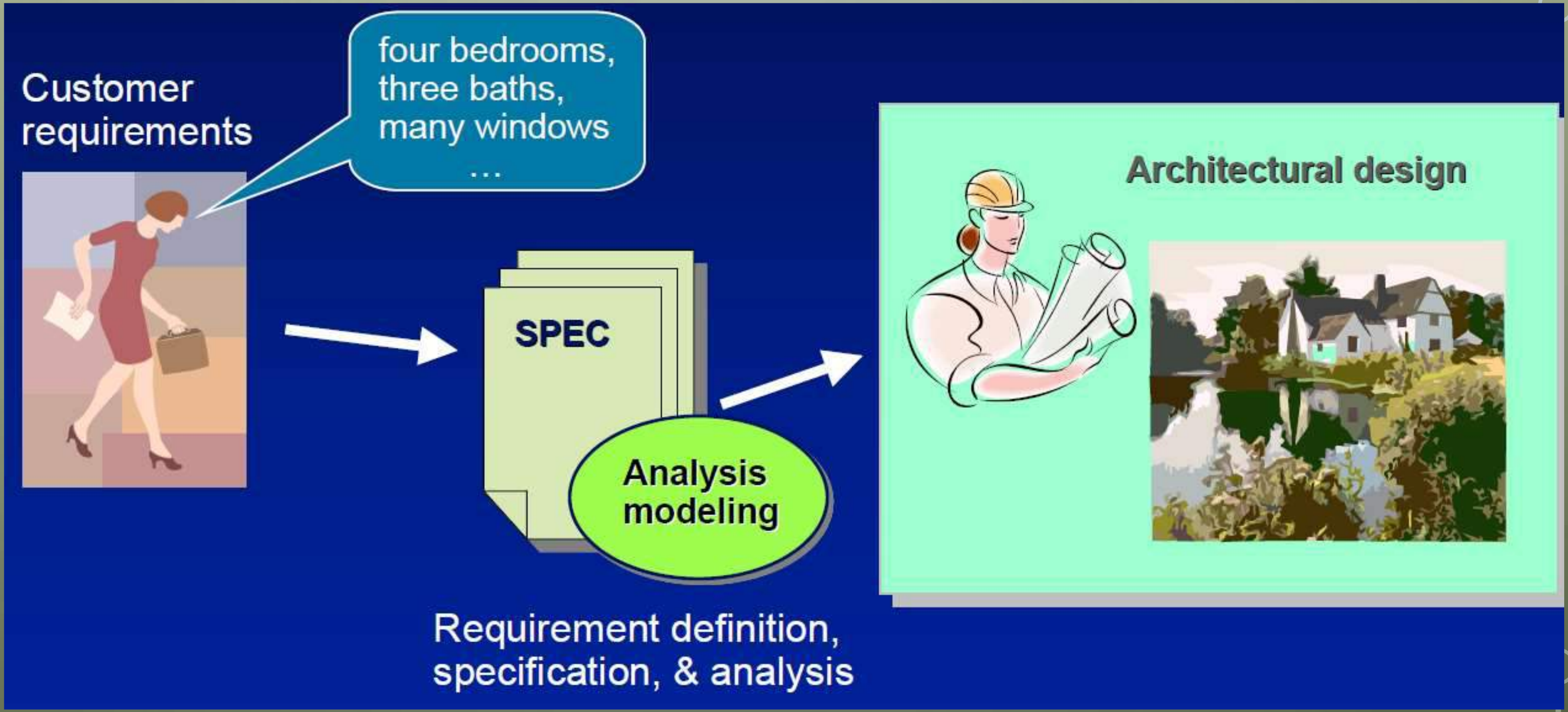
The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution - IEEE 1471:2000.

Source : http://www.iso-architecture.org/ieee-1471/defining-architecture.html

ARCHITECTURAL DESIGN

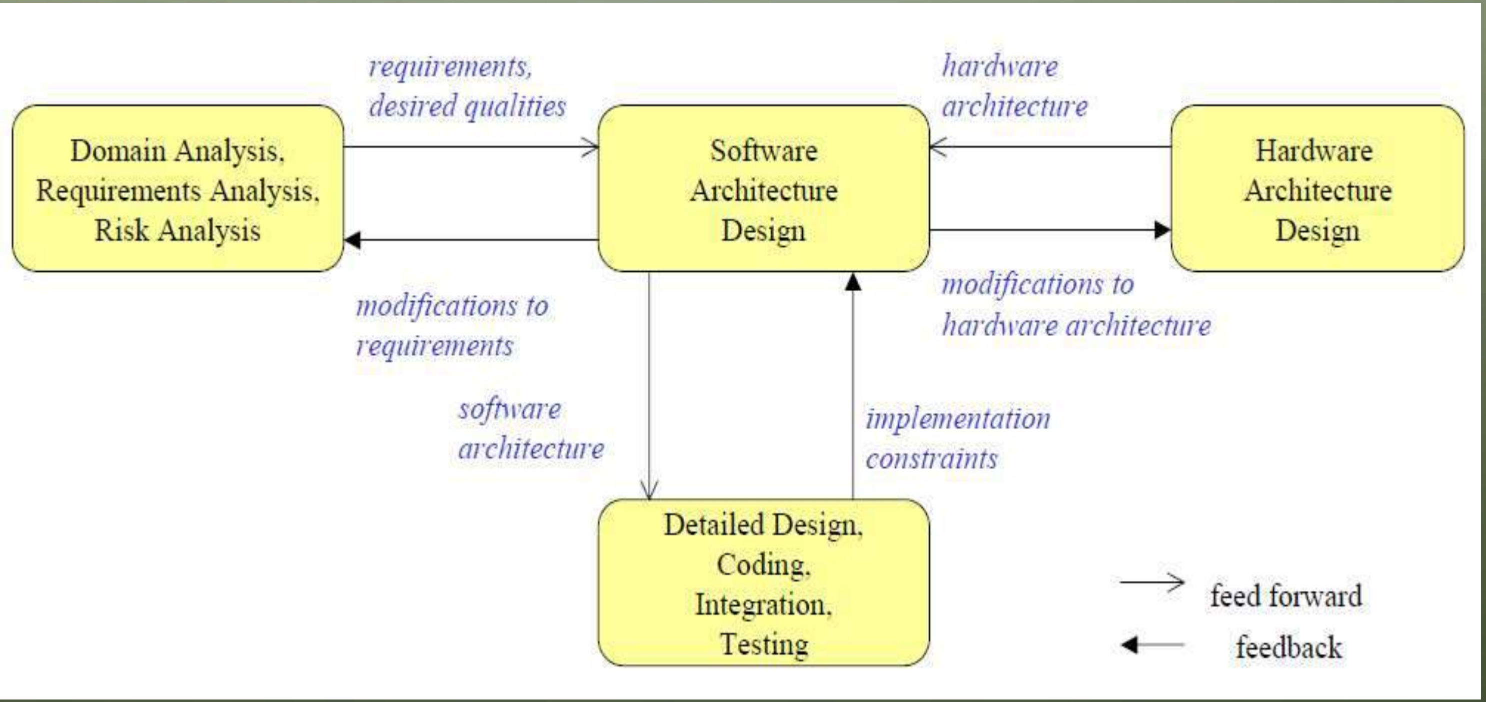
- An early stage of the system design process.
- High-level design (describe the decomposition of the software system into subsystems).
- Involves identifying major system components and their communications.
- Provides a design plan, a blueprint of a system, an abstraction to help manage the complexity of a system, and also a communication medium between stakeholders.

ARCHITECTURAL DESIGN



ARCHITECTURAL DESIGN

- Comes after the domain analysis, requirements analysis, and risk analysis, and before detailed design, coding, integration and testing.



ARCHITECTURAL DESIGN

- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- Separates the overall structure of the system, in terms of components and their interconnections, from the internal details of the individuals components.
- The output of this design process is a **description** of the software architecture.

ARCHITECTURAL DESIGN

- **Architecture in the small**
  - Concerned with the **architecture of individual programs**.
  - We are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large**
  - Concerned with the architecture of **complex enterprise systems** that include other systems, programs, and program components.
  - These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

WHY IS SOFTWARE 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.
  - Constraints on implementation
  - Dictates organizational structure
  - Inhibits or enable quality attributes

WHY IS SOFTWARE ARCHITECTURE IMPORTANT?

- Architecture is a **transferable abstraction** of a system
  - Product lines share a common architecture
  - Allows for template-based development
  - Basis for training (Foundation for training of a new team member)
- Reasoning about and **managing change**
  - The decisions made in an architecture allow you to reason about and manage change as the system evolves.

WHY IS SOFTWARE ARCHITECTURE IMPORTANT?

- **Predicting system qualities**
  - The analysis of an architecture enables early prediction of a system's qualities.
- Channels the **creativity of developers,** reducing design and system complexity.
- **Improving cost and schedule** estimates.
- Enabling evolutionary prototyping
  - can prototyped as skeletal system (at least some of the infrastructure is built before much of the system's functionality has been created.

SOFTWARE ARCHITECTURE CONTEXTS

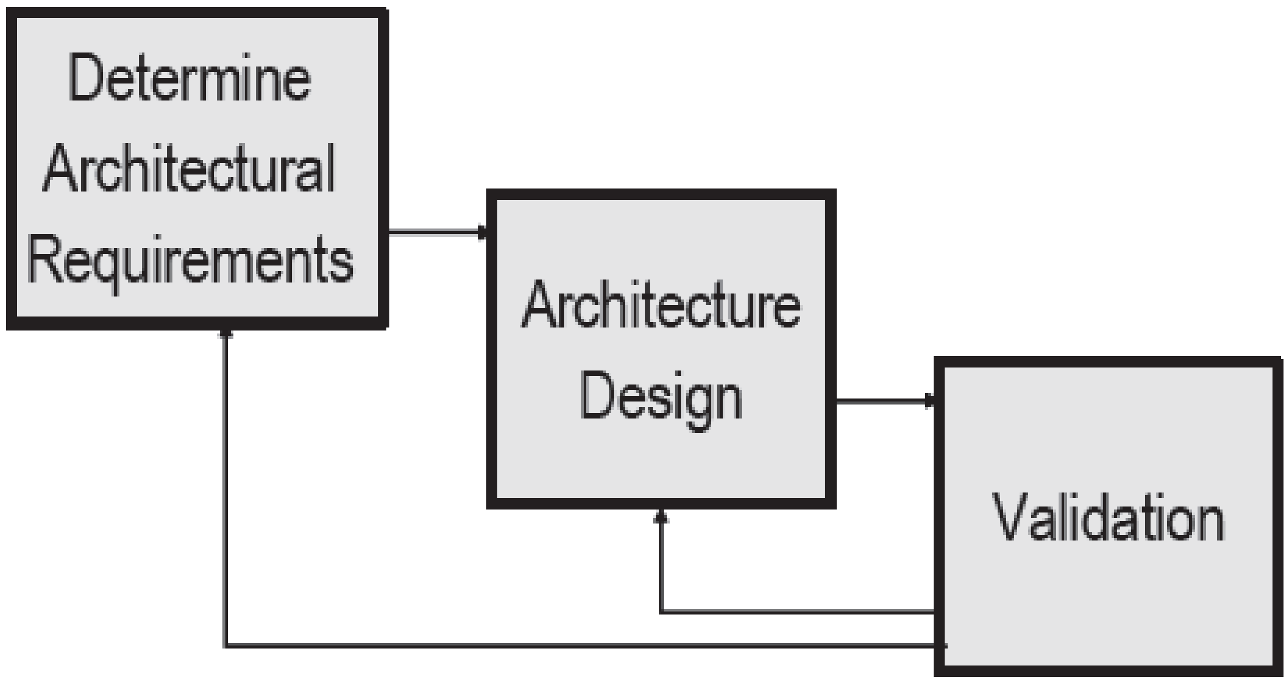
- **Technical** contexts
  - Includes the achievement of quality attribute requirements
  - Also includes the current technology such as cloud and mobile computing
- **Project life cycle** contexts
  - Regardless of the software development methodology you use, you must:
    - make a business case for the system
    - understand the architecturally significant requirements
    - create or select the architecture
    - document and communicate the architecture
    - analyze or evaluate the architecture
    - Implement and test the system based on the architecture
    - Ensure the implementation conforms to the architecture

SOFTWARE ARCHITECTURE CONTEXTS

- **Business** contexts
  - The system created from the architecture must satisfy the business goals of a wide variety of stakeholders, each of whom has different expectations for the system
- **Professional** contexts
  - You must have certain skills and knowledge to be an architect, and there are certain duties that you must perform as an architect

ARCHITECTURE PROCESS

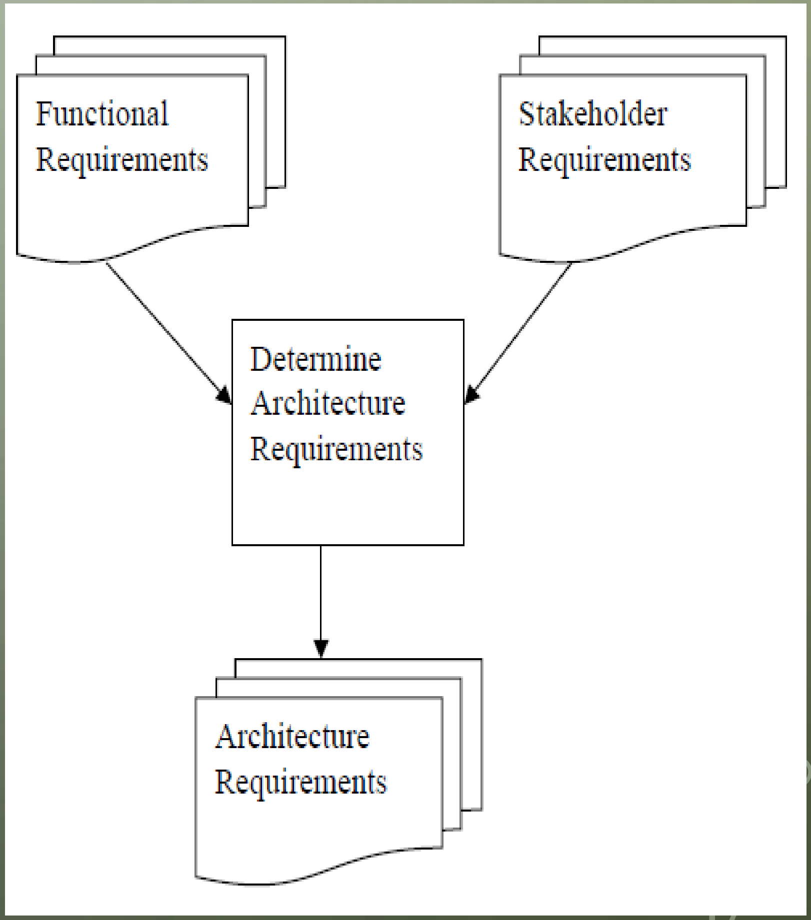
- Highly iterative
- Can scale to small/large projects



A three step architecture design process

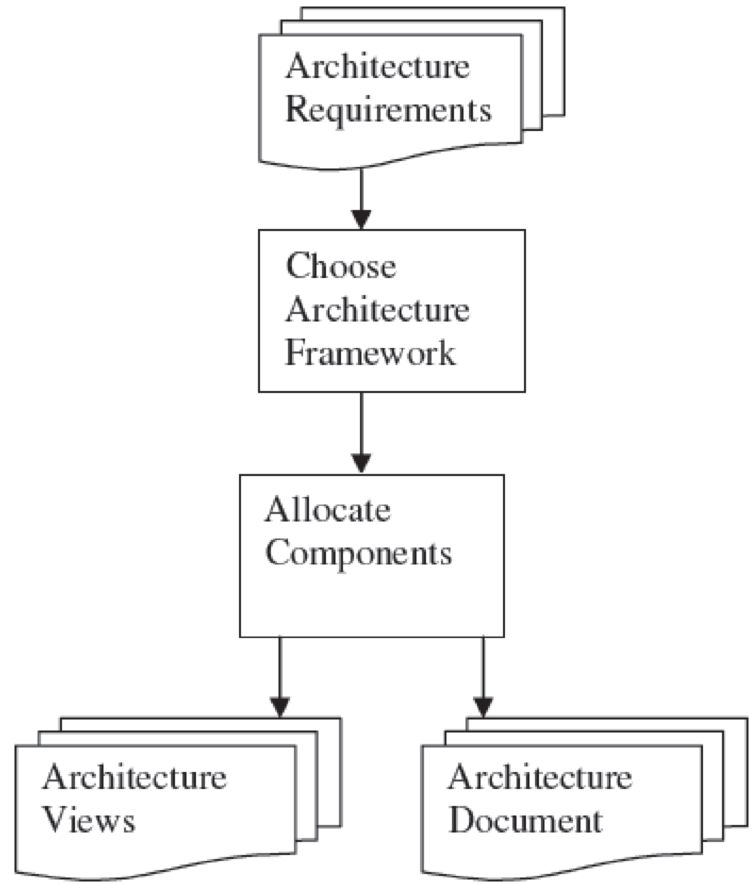
ARCHITECTURE PROCESS

- Determine Architectural Requirements
  - Architecture requirements sometimes called Architecturally Significant Requirements (ASR) or Architecture Use Cases.
  - Essentially the quality and non-functional requirements for the application.



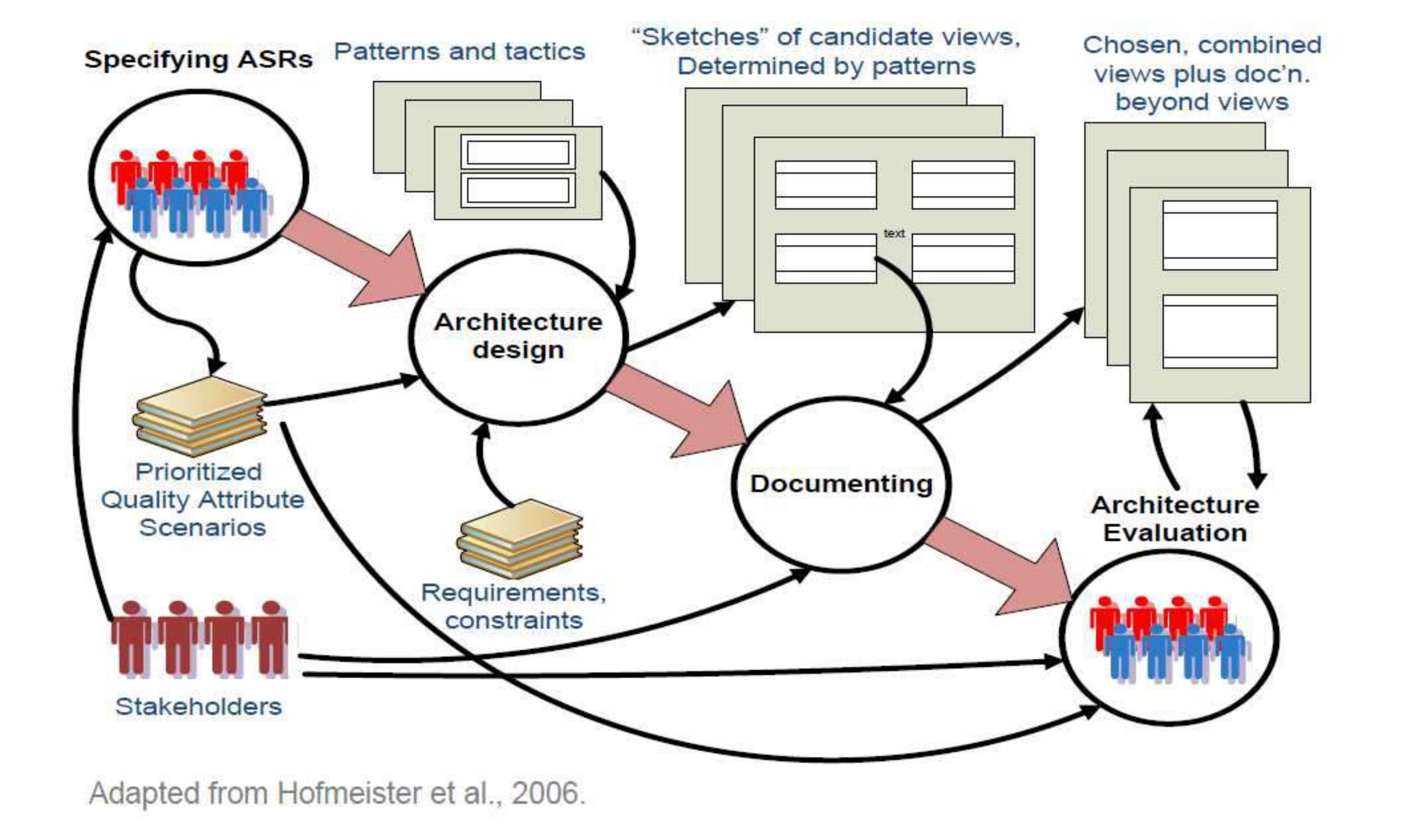
ARCHITECTURE PROCESS

- Inputs and outputs of architecture design

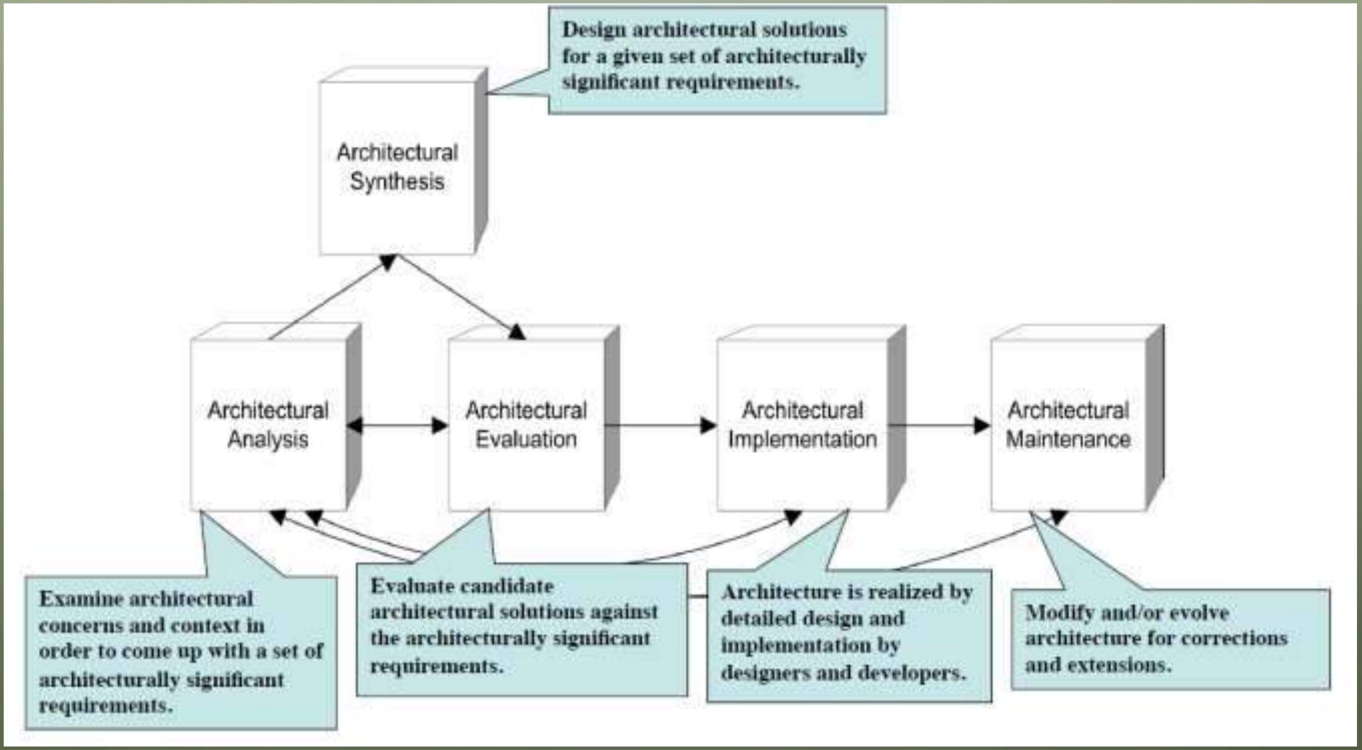


Inputs and outputs of architecture design

KEY ACTIVITIES IN SOFTWARE ARCHITECTURE PROCESS



ARCHITECTURE LIFE CYCLE



IN CLASS ACTIVITIES I

- Form groups and try to answer the following questions:
  - Compare and contrast software programming, software engineering and software design.
  - What are the roles required in a typical software engineering project?
  - Where does Software Design fit in the entire software engineering process?
  - What is a good design?
  - Does good design always yield good product?

DETAILED DESIGN

DETAILED DESIGN

- Once high-level design is done, you have:
  - a graphical representation of the structure of your software system
  - a document that defines high-level details of each module in your system, including
    - a module's interface (including input and output data types)
    - notes on possible algorithms or data structures the module can use to meet its responsibilities
    - a list of any non-functional requirements that might impact the module

DETAILED DESIGN

- After high-level design, a designer's focus shifts to **low-level design**
  - Each module's responsibilities should be specified as precisely as possible
  - Constraints on the use of its interface should be specified
  - pre and post conditions can be identified
  - module-wide invariants can be specified
  - internal data structures and algorithms can be suggested

DETAILED DESIGN

- However, a designer must exhibit caution to **not over specify** a design
    - as a result, we do not want to express a module's detailed design using a programming language
        - you would naturally end up implementing the module perhaps unnecessarily constraining the approaches a developer would use to accomplish the same job
    - nor do we (necessarily) want to use only natural language text to specify a module's detailed design
        - natural language text slips too easily towards ambiguity

DETAILED DESIGN

- Begins after the software architecture is specified, reviewed, and deemed sufficiently complete.
- Deals with the implementation part of what is seen as a system and its sub-systems.
- More detailed towards modules and their implementations.
- It defines logical structure of each module and their interfaces to communicate with other modules

DETAILED DESIGN

- Describes the desired behavior of the components interacting with each other.
- Focus on functional requirements.
- Two major tasks:
  - **Interface design**
    - Define the interfaces between system components.
  - **Component design**
    - Take each system component and design how it will operate.
  - Includes **Database design –** design the system data structures and how to be represented in database.

DETAILED DESIGN GOALS

- Create detailed "plans" (like blueprints) for implementation.
  - Designs are **blue-prints for code construction**
  - Important: A design should fully describe how coders will implement the system in the next phase
- Build these from requirements models so we are confident that all user needs will be met
- Create design models before coding so that we can:
  - Compare different possible design solutions
  - Evaluate efficiency, ease of modification, maintainability, etc.

DETAILED DESIGN GOALS

- Qualities Of A Good Design:
  - Correctness
    - It Should Lead To A Correct Implementation
  - Completeness
    - It Should Do Everything. Everything? It should follow the specifications.
  - Changeable
    - It Should Facilitate Change—Change Is Inevitable
  - Efficiency
    - It Should Not Waste Resources. But:
    - Better A Working Slow Design Than A Fast Design That Does Not Work
  - Simplicity
    - It Should Be As Understandable As Possible

SOFTWARE DESIGN PROCESS

- The software specifications are transformed into design models that describe the details of the data structures, systems architecture, interface and components.
- Each design product is reviewed for quality before moving to the next phase of software development.
- Concerned with describing ***how*** a requirement is to be met by the design product.

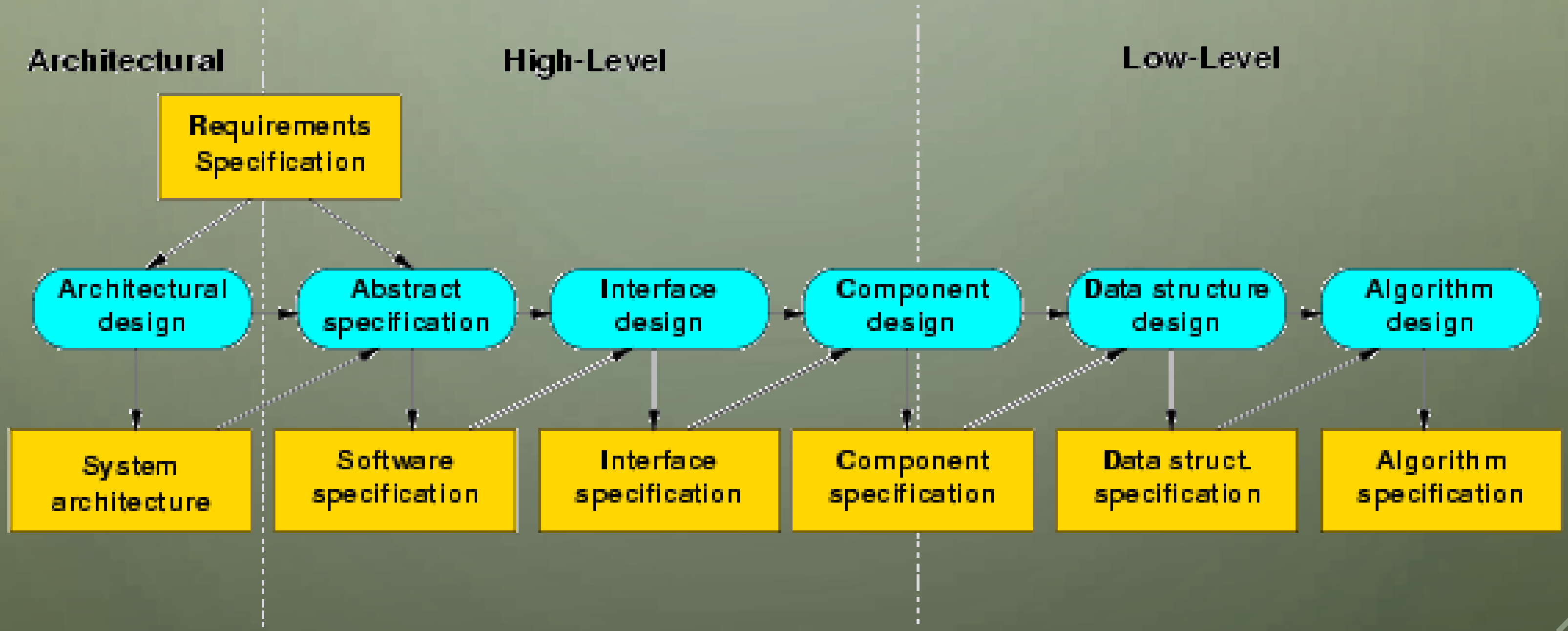
SOFTWARE DESIGN PROCESS

- Can be perceived as series of well-defined steps.
- Steps involved:
  - A solution design is created from requirement or previous used system and/or system sequence diagram.
  - Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
  - Class hierarchy and relation among them is defined.
  - Application framework is defined.

SOFTWARE DESIGN PROCESS

- At the end of the design process, a design specification document, also known as **Software Design Document (SDD)** is produced.
- This document is composed of the design models that describe the data, architecture, interfaces and components.

SOFTWARE DESIGN PROCESS ACTIVITIES(PHASES)



SOFTWARE DESIGN PROCESS ACTIVITIES(PHASES)

- *Architectural design*
  - The sub-systems making up the system and their relationships are identified and documented.
- Abstract specification
  - For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.
- Interface design
  - For each sub-system, its interface with other sub-system is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.

SOFTWARE DESIGN PROCESS ACTIVITIES(PHASES)

- *Component design*
  - Decompose sub-systems into components.
  - Services are allocated to different components and the interfaces of these components are designed.
- Data Structure design
  - The data structures used in the system implementation are designed in detail and specified (to hold problem data).
- Algorithm design
  - The algorithms used to provide services (for problem functions) are designed in detail and specified.

SOFTWARE PROCESS MODELS

- Generic Process Models
  - describe the organization of software processes.
  - Such as waterfall model, evolutionary and reuse-component.
- Iterative Process Models
  - describe the software process as a cycle of activities.
  - Such as incremental delivery and spiral model.
  - The main advantage is to avoid the commitment to specification and design.

*** Please read more about the process models on your own!!**

SOFTWARE DESIGN APPROACHES

- **Top Down Design**
  - Takes the whole software system (high level of abstraction) as one entity and then decomposes it into a more detailed level (lower level of abstraction) to achieve more than one sub-system or component based on some characteristics.
  - More suitable when the software solution needs to be designed from scratch and specific details are unknown.

SOFTWARE DESIGN APPROACHES

- **Bottom-up Design**
  - Starts with most specific and basic components.
  - It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.
  - More suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Applicable on most design projects; therefore, their use is expected to help achieve high-quality designs.
- Become fundamental drivers for decision making during the software design process.
- The principles include:
  - Modularization
  - Abstraction
  - Encapsulation (information hiding)
  - Coupling and cohesion
  - Separation of interface and implementation
  - Sufficiency and completeness

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Modularization
  - Divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software.
  - Usually the goal is to place different functionalities and responsibilities in different components

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Abstraction
  - Deals with creating conceptual entities required to facilitate problem solving by focusing on essential characteristics of entities while deferring (hold) unnecessary details.
  - At software architecture level – helps during identification of software components and their interfaces.
  - At detailed design phase – helps identify the entities, functions, and interfaces required to realize the component's provided services.
  - At construction level – helps further design of functions identified during detailed design.

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Encapsulation (Information hiding)
  - Information hiding is a decomposition principle that requires that each module hides its internal details and is specified by as little information as possible
    - Forces design units to communicate only through well-defined interfaces
    - Enables clients to be protected if internal details change
  - Enforces entities to communicate between each other using a “need to know only” basis.

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Coupling and Cohesion
  - Coupling
    - The manner and degree of interdependence between software modules.
    - The degree to which the modules of a design are related.
    - Three common types:
      - Content coupling – refers to modules that modify or rely on the internal details of other modules
      - Common coupling – refers to dependencies based on a common access area, such as a global variable.
      - Data coupling – refers to type of dependency in which design units communicate with each other only through a set of data parameters.

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Cohesion
  - The manner and degree to which the tasks performed by a single software module are related to each other.
  - Can be classified as:
    - Functional cohesion – all tasks in a design unit contribute to perform a single function.
    - Procedural cohesion – its tasks work procedurally (in steps) to achieve unit's purpose
    - Temporal cohesion – all tasks in a design unit are performed at specific times.
    - Communication cohesion – its tasks produce or consume the same data.

- **The ideal system has highly cohesive modules that are loosely coupled**

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Separation of Interface and Implementation
  - Involves defining a component by specifying a public interface (known to the clients) that is separate from the details of how the component is realized.
  - Should not be confused with encapsulation
  - Separating interface from implementation through concepts like **polymorphism** allows you to create several implementations of the same interface that do similar things in different ways.
  - Helps with code reuse.

SOFTWARE DESIGN CONCEPTS/PRINCIPLES

- Sufficiency and completeness
  - Means ensuring that a software component captures all the important characteristics of an abstraction and nothing more.
  - Completeness – a characteristic that measures how well design units provide required services to achieve their intent.
  - Sufficiency – measures how well design units are at providing only the services that are sufficient for achieving their intent.

CONSTRUCTION DESIGN

CONSTRUCTION DESIGN

- Some projects allocate considerable design activity to construction.
- While others allocate design to a phase explicitly focused on design.
- Regardless of the exact allocation, ***some detailed design work will occur at the construction level.***
- And that design work tends to be dictated by constraints imposed by the real-world problem that is being addressed by the software.

CONSTRUCTION DESIGN

- Just as construction workers building a physical structure must make ***small-scale modifications*** to account for unanticipated gaps in the builder's plans
- Software construction workers must make modifications on a smaller or larger scale to flesh out details of the software design during construction.
- Construction designs are applied on a smaller scale of ***algorithms, data structures, and interfaces.***

SUMMARY

- This chapter presented the fundamental of software design.
- It described the process involved in software design.
- It also described software design principles such as modularization, abstraction and encapsulation that can be used in software design.

IN CLASS ACTIVITIES II

- Form groups and try to answer the following questions:
  - Come up with usage scenarios for top-down and bottom-up design approaches
  - If a customer has no idea what he/she wants, how should we start the design process?
  - If a group of customers have no idea about what they want, how should we start the design process?
  - How should we handle customer requests for change before/during/after which the design process has started?