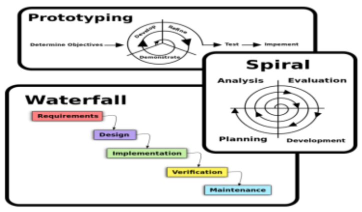
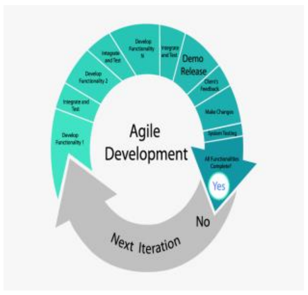
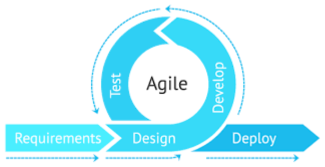


Software Engineering
Software Architecture





Outline

- Importance of Architecture
- Architecture Description & Decisions
- Architecture Mapping using data flow

2

Software Architectural & Design

- Architectural design represents the **structure of data and program components**
- **It considers**
  - **Architectural style** that the system will take
  - **Structure** and **properties** of the **components** that constitute the system
  - **Interrelationships** that occur among all architectural components of a system
- Representations of software architecture are an **enabler for communication between all parties** (stakeholders).
- Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together"

3

Software Architectural & Design

Marry in haste repent at leisure
True for Software Architecture Design

4

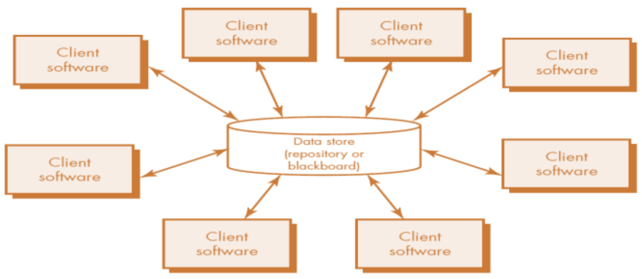
Architectural Styles

- Data-centered architecture style
- Data-flow architectures
- Call and return architecture
- Object-oriented architecture
- Layered architecture
- Each style describes a system category that encompasses
- A **set of components** (Ex., a database, computational modules) that **perform a function** required by a system.
- A set of **connectors** that **enable "communication, coordination and cooperation"** among components.
- **Constraints** that define **how components can be integrated** to form the system.
- **Semantic models** that enable a designer **to understand the overall properties** of a system.

5

Data-centered architecture style

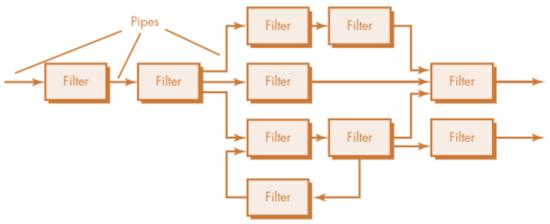
- **A data store** (Ex., a file or database) **resides at the center** of this architecture and is **accessed frequently** by other components.
- Client **software accesses a central repository.**
- In **some cases** the data **repository is passive**
- That is, client software accesses the data independent of any changes to the data or the actions of other client software.



6

Data-flow architecture

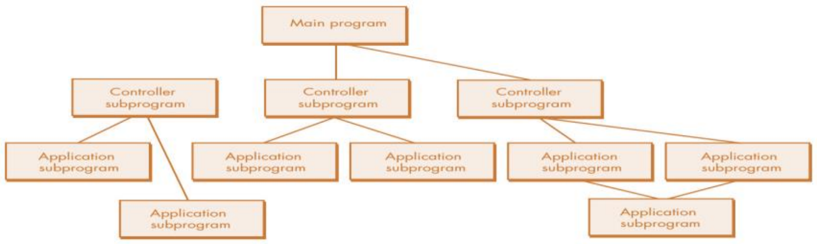
- This architecture is applied **when input data are to be transformed**.
- A set of components (called **filters**) **connected by pipes** that **transmit data** from one component to the next.
- Each filter works independently of those components upstream and downstream, is designed to
  - expect data input of a certain form, and
  - produces data output (to the next filter) of a specified form.



Pipes and filters

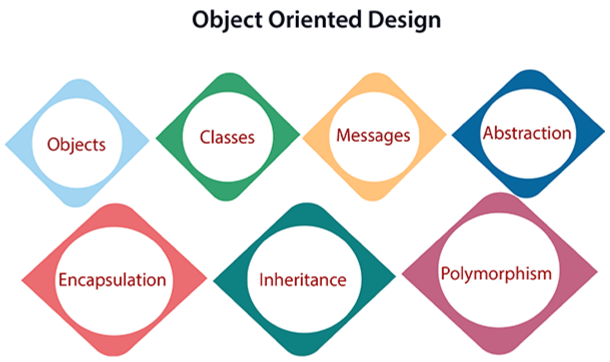
Call and return architecture

- This architectural style **enables a software designer (system architect) to achieve a program structure** that is relatively easy to modify and scale.
- A number of sub styles exist within this category as below.
  1. **Main program/subprogram architectures :** This classic program structure **decomposes function into a control hierarchy** where a **"main" program invokes a number of program components**, which in turn may invoke still other components.
  2. **Remote procedure call architectures :** The components of a main **program/subprogram** architecture are **distributed across multiple computers** on a network.



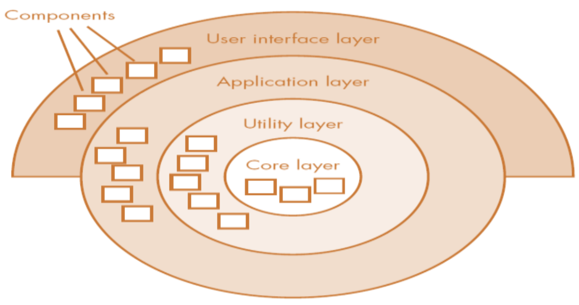
Object-oriented architecture

- The **components** of a system **encapsulate data and the operations** that must be applied to manipulate the data.
- **Communication** and **coordination** between components is accomplished **via message passing**.



Layered architecture

- A number of **different layers are defined**, each accomplishing **operations** that **progressively** become **closer to the machine instruction** set.
- At the **outer layer**, components service **user interface** operations.
- At the **inner layer**, components perform **operating system interfacing**.
- **Intermediate layers** provide **utility services** and application **software functions**.



Component Level or Procedural Design

Effective programmers should not waste their time in debugging, they should not introduce bug to start with

- Component is a modular, deployable and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.
- Component-level design occurs after data, architectural and interface designs have been established.
- It defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component.
- The intent is to translate the design model into operational software.
- But the abstraction level of the existing design model is relatively high and the abstraction level of the operational program is low.

11

Function Oriented Approach

- The following are the features of a typical function-oriented design approach:
- A system is viewed as something that performs a set of functions.
  - Starting at this high level view of the system, each function is successively refined into more detailed functions.
  - For example, consider a **function create-new-library member**
    - which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge.
    - This function may consist of the following sub-functions:
      - assign-membership-number, create-member-record and print-bill

12

Function Oriented Approach

        - Each of these sub-functions may be split into more detailed sub-functions and so on.
- The system **state is centralized** and **shared among different functions**.
    - For Ex., data such as **member-records** is available for reference and updating to several functions such as:
        - create-new-member
        - delete-member
        - update-member-record

13

Object Oriented Approach

- In the object-oriented design approach, the **system is viewed** as **collection of objects** (i.e., entities).
- The **state is decentralized** among the objects and **each object manages its own state** information.
- For example, in a Library Automation Software,
    - each library member may be a separate object with its own data and functions to operate on these data.
    - In fact, the functions defined for one object cannot refer or change data of other objects.
    - **Objects** have their **own internal data** which define their state.

14

Why Is Architecture Important?

- Three key reasons that software architecture is 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 model of how the system is structured and how its components work together"

Architecture Description

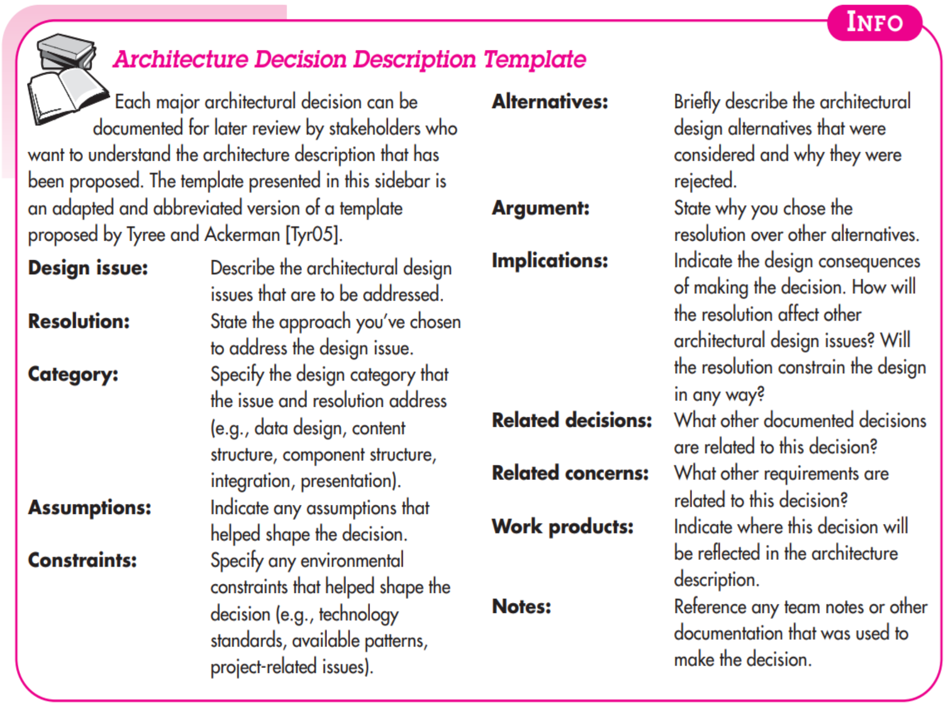
- Different stakeholders will see an architecture from different viewpoints that are driven by different sets of concerns.
- This implies that an architectural description is actually a set of work products that reflect different views of the system.
- "Developers want clear, decisive guidance on how to proceed with design. Customers want a clear understanding on the environmental changes that must occur and assurances that the architecture will meet their business needs. Other architects want a clear, salient understanding of the architecture's key aspects."
- Each of these "wants" is reflected in a different view represented using a different viewpoint.

Architecture Description

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems, [IEE00], with the following objectives:
  - to establish a conceptual framework and vocabulary for use during the design of software architecture,
  - to provide detailed guidelines for representing an architectural description, and
  - to encourage sound architectural design practices.

Architecture Decision

- Each view developed as part of an architectural description addresses a specific stakeholder concern.
- To develop each view (and the architectural description as a whole) the system architect considers a variety of alternatives and ultimately decides on the specific architectural features that best meet the concern.
- Therefore, architectural decisions themselves can be considered to be one view of the architecture. The reasons that decisions were made provide insight into the structure of a system and its conformance to stakeholder concerns.



INFO

Architecture Decision Description Template

Each major architectural decision can be documented for later review by stakeholders who want to understand the architecture description that has been proposed. The template presented in this sidebar is an adapted and abbreviated version of a template proposed by Tyree and Ackerman [Tyr05].

Design issue:	Describe the architectural design issues that are to be addressed.
Resolution:	State the approach you've chosen to address the design issue.
Category:	Specify the design category that the issue and resolution address (e.g., data design, content structure, component structure, integration, presentation).
Assumptions:	Indicate any assumptions that helped shape the decision.
Constraints:	Specify any environmental constraints that helped shape the decision (e.g., technology standards, available patterns, project-related issues).
Alternatives:	Briefly describe the architectural design alternatives that were considered and why they were rejected.
Argument:	State why you chose the resolution over other alternatives.
Implications:	Indicate the design consequences of making the decision. How will the resolution affect other architectural design issues? Will the resolution constrain the design in any way?
Related decisions:	What other documented decisions are related to this decision?
Related concerns:	What other requirements are related to this decision?
Work products:	Indicate where this decision will be reflected in the architecture description.
Notes:	Reference any team notes or other documentation that was used to make the decision.

Architecture Mapping using Data flow

- A comprehensive mapping that accomplishes the transition from the requirements model to a variety of architectural styles does not exist, and the designer must use the translation of requirements to design for these styles
- A mapping technique, called structured design, is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram.

Architecture Mapping using Data flow

- The transition from information flow (represented as a DFD) to program structure is accomplished as part of a six step process:

1. the type of information flow is established,
2. flow boundaries are indicated,
3. the DFD is mapped into the program structure,
4. control hierarchy is defined,
5. the resultant structure is refined using design measures and heuristics,
6. the architectural description is refined and elaborated.

Transform Mapping

- **Step 1. Review the fundamental system model**:
  - The fundamental system model or context diagram depicts the system function as a single transformation, representing the external producers and consumers of data that flow into and out of the function.
- **Step 2. Review and refine data flow diagrams for the software:**
  - Information obtained from the requirements model is refined to produce greater detail.
  - For example, the level 1 DFD.
  - The different level of data flow diagram exhibits relatively high cohesion.

Transform Mapping

- **Step 3. Determine whether the DFD has transform or transaction flow:**
  - characteristics. Evaluating the DFD to see data entering the software along which incoming path and exiting along which outgoing paths.
  - Therefore, an overall transform characteristic will be assumed for information flow.
- **Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries:**
  - Incoming data flows along a path in which information is converted from external to internal form; outgoing flow converts internalized data to external form.

Transform Mapping

- **Step 5. Perform “first-level factoring”:**
  - The program architecture derived using this mapping results in a top-down distribution of control.
  - Factoring leads to a program structure in which top-level components perform decision making and low level components perform most input, computation, and output work.
  - Middle-level components perform some control and do moderate amounts of work.
  - When transform flow is encountered, a DFD is mapped to a specific structure (a call and return architecture) that provides control for incoming, transform, and outgoing information processing.

Transform Mapping

- **Step 6. Perform "second-level factoring":**
  - Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture.
  - Beginning at the transform center boundary and moving outward along incoming and then outgoing paths, transforms are mapped into subordinate levels of the software structure.
- **Step 7. Refine the first-iteration architecture using design heuristics for improved software quality:**
  - A first-iteration architecture can always be refined by applying concepts of functional independence.