Chapter 6:

Architectural design:

Definition: understanding how a software system should be organized and designing the overall structure by decomposing system into various components

critical link between design and requirements engineering because identifies main structural components in a system and their relationships

Output is architectural model (set of communicating components)

Agility and architecture

overall systems architecture are made at the start of agile processes

refactoring this is expensive because can affect many components

Architectural abstraction

Architecture in the small: architecture of individual programs

Architecture in the large: architecture of complex enterprise systems distributed over different computers (managed by different companies)

Advantages of explicit architecture

- may be used as a focus of discussion by system stakeholders
- analysis of whether the system can meet non-functional requirements, can be done
- may be reusable across a range of systems or for many other products

Architectural representations

Most common method: Simple, informal block diagrams showing entities and relationships

But discontinued because:

- do not show the types of relationships between entities
- do not show visible properties of entities in the architecture

Box and line diagrams

Very abstract but useful for communication with stakeholders and project planning

Use of architectural models

- useful for communication with system stakeholders and project planning because provide abstract view of the system
- way of documenting as it can show all components, their interfaces and connections with other components

Architectural design decisions

some are common, but mostly depends on the type of system being developed.

affect non-functional characteristics of system

Architecture reuse

for systems of same domain

mostly built around a core architecture, that captures the essence. Hence, there can be multiple ways in which this core can be reused.

Architecture and system characteristics

Coarse-grained systems consist of fewer, larger components, fine-grained description regards smaller components of which the larger ones are composed

If requirment is:

- Performance then use coarse grained to decrease communication b/w components
- Security then layered architecture with critical assets in inner layers
- Safety then localize safety features in single or small components
- Availability then include redundant components for fault tolerance.
- Maintainability then use fine grain components

Architectural views

shows one view of the system

May show:

- decomposition of systems into modules
- interaction between run-time processes

for documentation multiple such views need to be shown

5 views: (4+1)

4+1 because use case also necessary for communication with stake holders

- logical : shows system functionalities and the components
- development : libraries, tools to be used. Project management budget, scheduling, Project manager task discussed here.
- physical : interface, OS to be used, deployment of system, what hardware to be deployed on, distributed or centralized (deployment diagram)
- process : process view and their interaction, also covers non functional requirements (sequence diagram)
- use case, user stories and scenarios is also an important view

Representing architectural views

UML used for describing and documenting system architectures however, not appropriate because does not cover abstractions appropriate for high-level system description.

Architectural description languages (ADLs) developed but not used widely

Architectural patterns

1. **MVC pattern:**

   Model: manages the system data and associated operations on that data. (Database)

   View: defines and manages how the data is presented to the user (Interface, forms)

Controller: manages user interaction(user click, key presses) and passes these interactions to the View and the Model (data validation, logic)

Used when:

- when multiple ways to interact with data
- when future requirements are known

Advantages:

- data can be changed independtly of represetation
- supports presentation of same data

Disadvantages:

- involves additional code and code complexity increases

2. **Layered architecture**

system into a set of layers, each providing a service

Supports the incremental development, on change only adjacent layers are affected

Used when:

- when building new facilities on top of existing systems
- when the development is spread across several teams
- requirement for multi-level security

Advantages:

- redundant facilities can be provided, increases dependibility of system
- easy replacement of layers

Disadvantages:

- Performance can be a problem because of multiple levels of interpretation
- providing seperation between layers is difficult

3. **Repository Pattern**
   All data in a central repository that is accessible to all system components. Communicate only through repository

   Used when:
   - when large volumes of information need to be stored for a long time
   - for data-driven systems, like weather based systems. Inclusion of data in the respository triggers an action

   Advantages:

   - Components can be independent, hence changes can be propagated to all components
   - All data can be managed consistently

   Disadvantages:

   - single point of failure
   - distribution of repositories across several computers can be difficult

4. **Client-server architecture**
   functionality of the system is organized into services, with each service delivered from a separate server to the client

   Used when:
   - when data in a shared database has to be accessed from a range of locations
   - when the load on a system is variable

   Advantages:
   - servers can be distributed across a network
   - All data can be managed consistently

Disadvantages:

- single point of failure, vulnerable to dos attacks
- management problems in case of servers owned by different organization
- performance may be unpredictable

5. **Pipe and filter architecture**
   each processing component (filter) is discrete and carries out one type of data transformation. The data flows from one component to another for processing

   Used when:
   - data processing applications (both batch- and transaction-based)

   Advantages:
   - transformations can be reused/added or evolved
   - can be implemented sequentially

   Disadvantages:

   - data format must be same across transformations, hence parsing and unparsing must be done for the input and output causing overhead
   - maybe impossible to use transformations due to incompatible data formats

Application architectures

designed to meet an organisational need.

Businesses are mostly common, so are their architectures

Such a generic application architecture can be made to adapt to business requirement

Use of application architectures

- serve as design checklist (checking design done properly)
- way of organising the work for the dev team
- components reusability can be assessed easily

Examples of application types

- ✧ **Data processing applications**
  - ✧ data-driven systems that does batch processing on data without user intervention such as billing systems
- ✧ **Transaction processing applications**
  - ✧ crud based systems as bank management systems etc.
  - ✧ interactive systems that allow information in a database to be remotely accessed and modified by a number of users
- ✧ **Event processing systems**
  - ✧ systems that depend on data gained from the environment as software for self driving cars
- ✧ **Language processing systems**
  - ✧ systems where user writes programs in a specified language processed and interpreted by the systems such as compilers

    OR
  - ✧ used to translate texts from one language into another and to carry out the instructions specified in the input language

Widely used application types:

- **Transaction processing systems**
  - ecommerce systems
  - reservations systems
- **Language processing systems**
  - compilers

    - interpreters

Some variations of transaction processing systems:

- **Information system architecture (transaction + layered approach):**
    - generic architecture consisting of:
        - user interface
        - user communications (authentication/login)
        - information retrieval (report generation, viewing data)
        - system database (transaction management) (create, update ,delete)
- **Web-based information systems(transaction + client-server):**
    - web server is responsible for all user communications by UI
    - application server is responsible for implementing logic as information retrieval (controller logic)
    - database server for crud operations

Language processing systems:

Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data.

Components:

- lexical analyzer : scans the source code and group characters into tokens
- symbol table : holds information about entities(variables, class names, object names)
- syntax analyzer : checks the syntax of the language being translated
- syntax tree : an internal structure made by syntax analyzer representing the program
- semantic analyzer : uses information from the syntax tree and the symbol table to check the semantic correctness

- code generator : traverses the syntax tree and generates abstract machine code

can use both repository architecture where syntax tree and symbol table stored in repository to be accessed by the analyzers and code generators

or can use pipeline method where code goes through the pipeline (lexical → syntax → semantic) and subsequent symbol table and syntax tree is generated etc.