PRINCIPLES OF SOFTWARE ENGINEERING

SECTION-C (NOTES)

SEM-6TH

ARCHITECTURAL DESIGN:

Software architecture serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams. The architecture is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that a design approach will yield an acceptable system. By building effective architecture, you can identify design risks and mitigate them early in the development process.

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

Why is Architecture Important?

Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals. For each of these areas, you should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.

The Goals of Architecture

Application architecture seeks to build a bridge between business requirements and technical requirements by understanding use cases, and then finding ways to implement those use cases in the software. The goal of architecture is to identify the requirements that affect the structure of the application. Good architecture reduces the business risks associated with building a technical solution. A good design is sufficiently flexible to be able to handle the natural drift that will occur over time in hardware and software technology, as well as in user scenarios and requirements. An architect must consider the overall effect of design decisions, the inherent tradeoffs between quality attributes (such as performance and security), and the tradeoffs required to address user, system, and business requirements.

Keep in mind that the architecture should:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

Key Architecture Principles

Consider the following key principles when designing your architecture:

- **Build to change instead of building to last**. Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.
- **Model to analyze and reduce risk**. Use design tools, modeling systems such as Unified Modeling Language (UML), and visualizations where appropriate to help you capture requirements and architectural and design decisions, and to analyze their impact. However, do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.
- **Use models and visualizations as a communication and collaboration tool**. Efficient communication of the design, the decisions you make, and ongoing changes to the design, is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share your design efficiently with all the stakeholders, and to enable rapid communication of changes to the design.
- **Identify key engineering decisions**. Use the information in this guide to understand the key engineering decisions and the areas where mistakes are most often made. Invest in getting these key decisions right the first time so that the design is more flexible and less likely to be broken by changes.

SYSTEM DESIGN:

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Data design is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data. For specifying and designing efficient data structures, some principles should be followed. These principles are listed below.

The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.

A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.

Language used for developing the system should support abstract data types.

The structure of data can be viewed at three levels, namely, program component level, application level, and business level. At the program component level, the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the application level, it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the business level, the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.

Data modeling in software engineering is the process of creating a data model for an information system by applying formal data modeling techniques.

Data modeling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modeling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modeling:

- to assist business analysts, programmers, testers, manual writers, IT package selectors, engineers, managers, related organizations and clients to understand and use an agreed semi-formal model the concepts of the organization and how they relate to one another
- to manage data as a resource
- for the integration of information systems
- for designing databases/data warehouses (aka data repositories)

Data modeling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in a repository so that they can be retrieved, expanded, and edited over time. Whitten et al. (2004) determined two types of data modeling:

- Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems is defined. Information engineering is a methodology that embraces this approach.
- Data modeling during systems analysis: In systems analysis logical data models are created as part of the development of new databases.

Data modeling is also used as a technique for detailing business requirements for specific databases. It is sometimes called *database modeling* because a data model is eventually implemented in a database

A **data model** is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities. For instance, a data model may specify that the data element representing a car be composed of a number of other elements which, in turn, represent the color and size of the car and define its owner.

Types of data models

Database model A database model is a specification describing how a database is structured and used.Several such models have been suggested. Common models include:

Flat model

This may not strictly qualify as a data model. The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another.

Hierarchical model

The hierarchical model is similar to the network model except that links in the hierarchical model form a tree structure, while the network model allows arbitrary graphs.

Network model

This model organizes data using two fundamental constructs, called records and sets. Records contain fields, and sets define one-to-many relationships between records: one owner, many members. The network data model is an abstraction of the design concept used in the implementation of databases.

Relational model

is a database model based on first-order predicate logic. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The power of the relational data model lies in its mathematical foundations and a simple user-level paradigm.

Object-relational model

Similar to a relational database model, but objects, classes and inheritance are directly supported in database schemas and in the query language.

Star schema

The simplest style of data warehouse schema. The star schema consists of a few "fact tables" (possibly only one, justifying the name) referencing any number of "dimension tables". The star schema is considered an important special case of the snowflake schema.

Data structure diagram

A data structure diagram (DSD) is a diagram and data model used to describe conceptual data models by providing graphical notations which document entities and their relationships, and the constraints that bind them. The basic graphic elements of DSDs are boxes, representing entities, and arrows, representing relationships. Data structure diagrams are most useful for documenting complex data entities.

Data structure diagrams are an extension of the entity-relationship model (ER model). In DSDs, attributes are specified inside the entity boxes rather than outside of them, while relationships are drawn as boxes composed of attributes which specify the constraints that bind entities together. The E-R model, while robust, doesn't provide a way to specify the constraints between relationships, and becomes visually cumbersome when representing entities with several attributes. DSDs differ from the ER model in that the ER model focuses on the relationships

between different entities, whereas DSDs focus on the relationships of the elements within an entity and enable users to fully see the links and relationships between each entity.

Entity-relationship mode

An entity-relationship model (ERM), sometimes referred to as an entity-relationship diagram (ERD), is an abstract conceptual data model (or semantic data model) used in software engineering to represent structured data. There are several notations used for ERMs.

Geographic data model

A data model in Geographic information systems is a mathematical construct for representing geographic objects or surfaces as data. For example,

- the vector data model represents geography as collections of points, lines, and polygons;
- the raster data model represent geography as cell matrixes that store numeric values;
- and the Triangulated irregular network (TIN) data model represents geography as sets of contiguous, non overlapping triangles.

Generic data model

Generic data models are generalizations of conventional data models. They define standardized general relation types, together with the kinds of things that may be related by such a relation type. Generic data models are developed as an approach to solve some shortcomings of conventional data models. For example, different modelers usually produce different conventional data models of the same domain. This can lead to difficulty in bringing the models of different people together and is an obstacle for data exchange and data integration. Invariably, however, this difference is attributable to different levels of abstraction in the models and differences in the kinds of facts that can be instantiated (the semantic expression capabilities of the models). The modelers need to communicate and agree on certain elements which are to be rendered more concretely, in order to make the differences less significant.

Semantic data model

A semantic data model in software engineering is a technique to define the meaning of data within the context of its interrelationships with other data. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. A semantic data model is sometimes called a conceptual data model.

DATABASE AND DATA WAREHOUSE

A **database** is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

A **database management system** (**DBMS**) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, MongoDB, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, MemSQL and IBM DB2. A database is not generally portable across different DBMSs, but different DBMS can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one DBMS. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language.[*disputed – discuss*] Sometimes a DBMS is loosely referred to as a 'database

In computing, a **data warehouse** (**DW** or **DWH**), also known as an **enterprise data warehouse** (**EDW**), is a system used for reporting and data analysis, and is considered a core component of business intelligence.[1] DWs are central repositories of integrated data from one or more disparate sources. They store current and historical data and are used for creating analytical reports for knowledge workers throughout the enterprise. Examples of reports could range from annual and quarterly comparisons and trends to detailed daily sales analysis.

The data stored in the warehouse is uploaded from the operational systems (such as marketing or sales). The data may pass through an operational data store for additional operations before it is used in the DW for reporting.

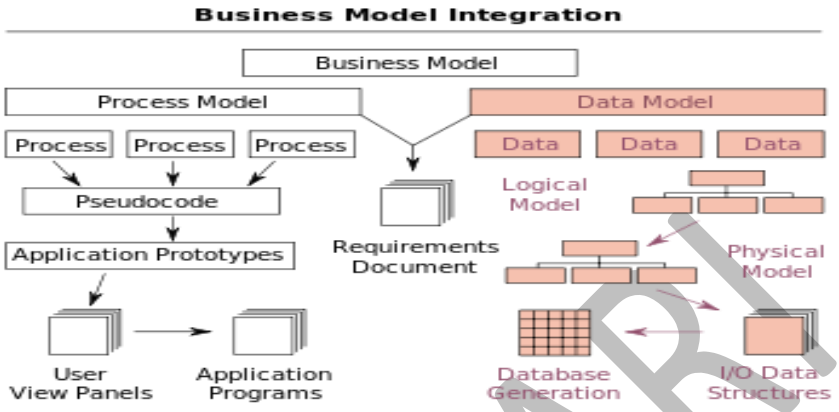
Types of systems

Data mart

A data mart is a simple form of a data warehouse that is focused on a single subject (or functional area), hence they draw data from a limited number of sources such as sales, finance or marketing. Data marts are often built and controlled by a single department within an organization. The sources could be internal operational systems, a central data warehouse, or external data. Denormalization is the norm for data modeling techniques in this system. Given that data marts generally cover only a subset of the data contained in a data warehouse, they are often easier and faster to implement.

Predictive analysis

Predictive analysis is about finding and quantifying hidden patterns in the data using complex mathematical models that can be used to predict future outcomes. Predictive analysis is different from OLAP in that OLAP focuses on historical data analysis and is reactive in nature, while predictive analysis focuses on the future. These systems are also used for CRM (customer relationship management).



Overview of data modeling context: Data model is based on Data, Data relationship, Data semantic and Data constraint. A data model provides the details of information to be stored, and is of primary use when the final product is the generation of computer software code for an application or the preparation of a functional specification to aid a computer software make-or-buy decision. The figure is an example of the interaction between process and data models.

A data model explicitly determines the structure of data. Data models are specified in a data modeling notation, which is often graphical in form.

Analysis of Alternatives

Any requirements analysis method combines a set of distinct heuristics and a unique notation to analyze information, function and behavior for a computer-based system. Through the application of the fundamental analysis principles, each method creates a model of the problem and the requirements for its solution.

Most requirements analysis methods are *information driven.* That is, the method provides a mechanism for representing the information domain of the problem. From this representation, function and behavior is derived and other software characteristics are established.

Requirements Analysis Methods

Requirements analysis methods enable an analyst to apply fundamental analysis principles in a systematic fashion.

Data Structure-Oriented Methods: Data structure-oriented analysis methods represent software requirements by focusing on data structure, rather than data flow. Although each data

structure-oriented method has a distinct approach and notation, all have some characteristics in common: (1) each assists the analyst in identifying key *information objects* (also called *entities* or *items*) and *operations* (also call *actions* or *processes*); (2) each assumes that the structure of information is hierarchical; (3) each requires that the data structure be represented using the sequence, selection and repetition constructs

Data Structured Systems Development

> *Data Structured Systems Development* (DSSD), also called the Warnier-Orr methodology, evolved from pioneering work on information domain analysis conducted by J. D. Warnier. Warnier developed a notation for representing information hierarchy using the three constructs for sequence, selection and repetition and demonstrated that the software structure could be derived directly from the data structure.

Software Complexity

Software complexity is a natural byproduct of the functional complexity that the code is attempting to enable. With multiple system interfaces and complex requirements, the complexity of software systems sometimes grows beyond control, rendering applications and portfolios overly costly to maintain and risky to enhance. Left unchecked, software complexity can run rampant in delivered projects, leaving behind bloated, cumbersome applications.

The software engineering discipline has established some common measures of software complexity. Perhaps the most common measure is the McCabe essential complexity metric. This is also sometimes called cyclomatic complexity. It is a measure of the depth and quantity of routines in a piece of code.

When measuring complexity, it is important to look holistically at coupling, cohesion, SQL complexity, use of frameworks, and algorithmic complexity. It is also important to have an accurate, repeatable set of complexity metrics, consistent across the technology layers of the application portfolio to provide benchmarking for continued assessment as changes are implemented to meet business or user needs. A robust software complexity measurement program provides an organization with the opportunity to:

- Improve Code Quality
- Reduce Maintenance Cost
- Heighten Productivity
- Increase Robustness
- Meet Architecture Standards

A structure chart is produced by the conversion of a DFD diagram; this conversion is described as ‘transform mapping (analysis)’. It is applied through the ‘transforming’ of input data flow into output data flow.

Transform analysis establishes the modules of the system, also known as the primary functional components, as well as the inputs and outputs of the identified modules in the DFD. Transform analysis is made up of a number of steps that need to be carried out. The first one is the dividing of the DFD into 3 parts:

Input

Logical processing

Output

The 'input' part of the DFD covers operations that change high level input data from physical to logical form e.g. from a keyboard input to storing the characters typed into a database. Each individual instance of an input s called an 'afferent branch'.

The 'output' part of the DFD is similar to the 'input' part in that it acts as a conversion process. However, the conversion is concerned with the logical output of the system into a physical one e.g. text stored in a database converted into a printed version through a printer. Also, similar to the 'input', each individual instance of an output is called as 'efferent branch'.

Transaction Flow

Similar to 'transform mapping', transaction analysis makes use of DFDs diagrams to establish the various transactions involved and producing a structure chart as a result.

Transaction mapping Steps:

Review the fundamental system model

Review and refine DFD for the SW

Assess the DFD in order to decide the usage of transform or transaction flow.

Identify the transaction center and the flow characteristics along each action path

Find transaction center

Identify incoming path and isolate action paths

In transaction mapping a single data item triggers one of a number of information flows that affect a function implied by the triggering data item. The data item implies a transaction.