1.1 INTRODUCTION TO SOFTWARE ENGINEERING

The Evolving Role of Software:

- Software can be considered in a dual role. It is a **product** and, a vehicle for delivering a product.
- As a product, it delivers the computing potential in material form of computer hardware.

  Ex: A network of computers accessible by local hardware, whether it resides within a cellular phone or operates inside a mainframe computer.
- As the vehicle, used to deliver the product. Software delivers the most important product of our time—**information**. Software transforms personal data; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.
- Software acts as the basis for operating systems, networks, software tools and environments.

Changing Nature of Software:

The following categories of computer software present the challenges for the software.

System software:

System software is a collection of programs written to service other programs.

Ex: Compilers, editors, and file management utilities. Operating system components, drivers, Telecommunications processors process largely indeterminate data.

Application Software:

Application Software consists of standalone programs that solve a specific business need.

Ex: Point of sale transaction processing, real time manufacturing process control.

Engineering and Scientific Software:

This is the software using "number crunching" algorithms for different science and applications.

System simulation, computer-aided design.

Embedded Software:

Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.

It has very limited and mysterious functions and control capability.

Product-line software:

It is designed to provide a specific facility for use by many different customers.

Ex: Inventory control products, word processing, spreadsheets, graphics, multimedia.

Web-based software:

The Web pages retrieved by a browser are software that includes executable instructions

Ex: HTML, Perl, or Java

Artificial Intelligence (AI) Software:

AI software makes use of non numerical algorithms to solve complex problems.

Active areas are expert systems, pattern recognition, games.

SOFTWARE MYTHS

Software myths will circulate misinformation and confusion, they appeared to be reasonable statements of fact (some times containing truths), but they had a sensitive feel;

They have a number of attributes that made them dangerous.

Old attitudes and habits are difficult to modify, and remnants of software myths are still believed.

Management myths:

Managers with software responsibility are often under pressure to maintain budgets, keep schedules, and maintain quality.

Myths:

- We already have a book that's full of standards and procedures for building software; won't that provide many people with everything they need to know?
- Many people have state-of-the-art software development tools; after all, we buy them the newest computers.
- If we get behind schedule, we can add more programmers and catch up.
- If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Customer myths:

Customers may be defined as follows;

- An outside company that has requested software under contract
- a person next to your desk
- a technical group
- a marketing or sales group

Customer myths lead to false expectations by the customer and ultimately there will be dissatisfaction with the developer.

Myths:

- A general statement of objectives is sufficient to begin writing programs and we can fill in the details later.
- Project requirements continually change, but change can be easily accommodated because software is flexible.

Practitioner's myths:

A Practitioner may be Planning group, Development group Verification group Support group, Marketing/sales.

Myths:

- Once we write the program and get it to work, our job is done.
- Until I get the program "running" I have no way of assessing its quality.
- The only deliverable work product for a successful project is the working program.
- The major task of a software engineer is to write a pr⸝gram.
- Schedule and requirements are the only importar ι things we should concern when we write programs.

1.3 A GENERIC VIEW OF PROCESS

What is software Engineering?

There are many definitions for software Engineering, a definition proposed by Fritz Bauer is;
"*Software engineering is the establishment and use of sound engineerin; principles in order to obtain economically software that is reliable and work efficiently on real machines*"

The IEEE definition for software Engineering is;

"**Software Engineering**:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

(2) The study of approaches as in the above statement"

Software Engineering: A layered technology:

Software engineering is a layered technology. It is divided into layers of different responsibilities. The layers present in software engineering are shown in the following diagram.

1.4 A PROCESS FRAMEWORK

A *common process framework* is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. Umbrella activities are independent of any one framework activity and occur throughout the process.

Frame work activities:

- A common process framework is established by defining a small number of framework activities.
- Software process framework can be used to all software projects. It consists of a number of *task sets*—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points—which enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.

Umbrella activities:

- Umbrella activities are such as software quality assurance, software configuration management, and measurement.
- Umbrella activities are independent of any one framework activity and occur throughout the process.
- There has been a significant emphasis on "process maturity."

1.5 THE CAPABILITY MATURITY MODEL INTEGRATION (CMM)

- The CMMI defines each process area in terms of "***specific goals***" and the "***specific practices***" required to achieve these goals.
- Specific goals: establish the characteristics that must exist if the activities implied by a process area are to be effective.
- Specific practices: refine a goal into a set of process related
- Software Engineering Institute (SEI) has developed a comprehensive model. To determine an organization's current state of process maturity,
- The SEI uses an assessment that results in a five point grading scheme. The grading scheme determines agreement with a 'capability maturity model (CMM)' that defines key activities required at different levels of process maturity.

Levels are defined in the following manner:

Level 0: **Incomplete:**

- The process area is either not performed or does note achieve all goals and objectives defined by the CMMI for level 1 capability

Level 1: Performed:

- All of the specific goals of the process area have been satisfied.
- Work tasks required to produce defined work products are being conducted.

Level 2: Managed:

- All level 1 criteria have been satisfied in addition all work associated with the process area conforms to an organizationally defined policy;
- All people doing the work have access to adequate resources to get the job done.

Level 3: Defined:

- All level 2 criteria have been achieved.
- In addition, the process is tailored from the organization's set of standard processes and contributes work products measures,

Level 4: Quantitatively managed:

- All level 3 criteria have been achieved;
- In addition, The process area is controlled and improved usin measurement and quantitative assessment

Level 5: Optimized:

- All capability level 4 criteria have been achieved:
- In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

PROCESS PATTERNS

A project's process specification defines the tasks the project should perform, and the order in which they should be done. Process patterns define a set of activities, actions, work tasks, work products and related behaviors that must be done to complete the project.

- A template is used to define a pattern

Typical examples are:

- Customer communication (a process activity)
- Analysis (an action)
- Requirements gathering (a process task)
- Reviewing a work product (a process task)
- Design model (a work prod

1.6 PROCESS MODELS

A **process model** specifies a general process, usually as a set of stages in which a project should be divided, the order in which the stages should be executed, and any other constraints and conditions on the execution of stages.

Select a process model is chosen based on:

- the nature of the project and application
- the methods and tools to be used
- the controls and deliverables

All software development can be characterized as a problem solving loop in which four distinct stages are encountered:

1. Status quo
2. Problem definition
3. Technical development
4. Solution integration

1. **Status quo:**

   It represents the current state of affairs

2. **Problem definition;**

   Identifies the specific problem to be solved and defines them

3. **Technical development:**

   Solves the problem through the application of some technology

4. **Solution integration:**

   It applies the solution and delivers the results e.g., documents, programs, data, new business function, new product

Evolutionary process models:

Business and product requirements often change as development proceeds, Classic process models are not designed to deliver a production system

Due to their assumptions on:

- A complete system will be delivered after the linear sequence is completed.
- Customer knows what they want at the early stage.

The realty in a software production process

- A lot of requirements changes during the production course
- A lot of iterative activities and work because of the evolutionary nature of software production

The several evolution process models proposed are:

- The incremental model
- The spiral model
- The component assembly model
- The concurrent development model

The Unified process:

- Unified process is an attempt to draw on the best features and characteristics of conventional software process models.
- It recognizes the importance of customer communication and makes more efficient methods for describing the customer's view of a system.

- The Unified process can be characterized as phases, they are

The phases of the unified process are

1. Inception
2. Elaboration
3. Construction
4. Transition
5. production

2.1 THE WATERFALL MODEL

This waterfall model is also termed as **linear sequential model**. Linear ordering of activities has some important consequences. First, to clearly identify the end of a phase and the beginning of the next, some certification mechanism has to be employed at the end of each phase. This is usually done by some verification and validation means that will ensure that the output of a phase is consistent with its input (which is the output of the previous phase), and that the output of the phase is consistent with the overall requirements of the system.

Waterfall model suggests a systematic, sequential approach to software development that begins at the system level and progresses through different phases like:

1. Analysis
2. Design
3. Coding
4. Testing
5. Support

1. Analysis:

- The requirements analysis is focused to understand the nature of the programs to be built.
- The software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface.
- In Requirement analysis the requirements for both the system and the software are documented and reviewed with the customer.

2. Design:

- Software design is actually a multi step process that focuses on data structure, software architecture, interface representations, and procedural detail.
- The design process translates requirements into a representation of the software that can be assessed for quality before coding begins.
- Like requirements, the design is documented and becomes part of the software configuration management.

3. Coding:

- The code generation step performs translation of design into a machine-readable form.
- If design is performed in a detailed manner, code generation can be accomplished mechanistically.
- The outputs of the earlier phases are often called ***work products*** and are usually in the form of documents like the requirements document or design document. For the coding phase, the output is the code.

4. Testing:

- Once code has been generated, program testing begins.
- The testing process focuses on the logical internals of the software
- It conducts tests for uncovered errors and ensures that defined input will produce actual results that agree with required results.

5. Support:

- Software will undergo change after it is delivered to the customer.
- Change will occur because errors have been encountered

e.g., a change required because of a new operating system

- Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Advantages:

- It is simple, step-by-step, focused, and easy to follow.
- It is straightforward and divides the large task of building a software system into a series of cleanly divided phases, each phase dealing with a separate logical concern.
- It is also easy to administer in a contractual setup—as each phase is completed and its work product produced, some amount of money is given by the customer to the developing organization.

Disadvantages:

- It assumes that the requirements of a system can be frozen (i.e., baselined) before the design begins. This is not possible for all systems.
- It follows the "big bang" approach—the entire software is delivered in one shot at the end. That is, it has the "all or nothing" value proposition.
- It is a document-driven process that requires formal documents at the end of each phase.
- Inflexible because real projects rarely follow the sequential flow that the model proposes.

- It is often difficult for the customers to state all requirements explicitly.
- The customer must have patience to wait to validate the software product in the late phases

2.2 INCREMENTAL PROCESS MODELS

There are two types of incremental models. These models develop the product in step by step incremental model. The product is developed to a little detail in first part and that is elaborated in the next phase. The popular incremental models are:

1. Prototyping Model:
2. The RAD Model:

Prototyping Model:

- The goal of a prototyping-based development process is to counter the limitations of the waterfall model. The basic idea here is that instead of freezing the requirements before any design or coding can proceed, a throw-away prototype is built to help understand the requirements. This prototype is developed based on the currently known requirements.
- A prototyping model may offer a best approach. It assists the software engineer and the customer to better understanding.
- It begins with communication Developer and customer meets and defines the overall objectives for the software
- A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user
- The quick scan leads to the construction of a prototype. The prototype is deployed and then evaluated by the customer/user.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

So totally we see the steps involved in prototype is

Step 1: Requirements gathering

Step 2: A “quick design” focuses on visible functions and behaviors of the product

Step 3: Prototype construction

Step 4: Customer evaluation of the prototype loop back

Advantages:

- Prototyping is well suited for projects where requirements are hard to determine and the confidence in the stated requirements is low.
- Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements.
- It is also an effective method of demonstrating the feasibility of a certain approach.
- Overall, in projects where requirements are not properly understood in the beginning, using the prototyping process model can be the most effective method for developing the software.

Disadvantages:

- Prototype can be problematic for the following reasons:

  The prototype can serve as "the first system". Brooks recommends we throw away.

  Developers usually attempt to develop the product based on the prototype.

  Developers often make implementation compromises in order to get a prototyping working quickly.

  Customers may be unaware that the prototype is not a product, which is held with.

- Prototyping is often not used, as it is feared that development costs may become large.

The RAD Model:

- ***Rapid application development*** (RAD) is an incremental software development process model that highlights an extremely short development cycle.
- The RAD model is a "high-speed" adaptation of waterfall model. in which rapid development is achieved by using component based construction
- Communication works is needed to understand the business problem and the information characteristics
- Planning is essential because multiple software teams work in parallel on different system functions
- Modeling includes five major phases
  1) **Business modeling:** Business modeling answers the questions like what information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?
  2) **Data modeling:** The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined.
  3) **Process modeling:** The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
  4) **Application generation:** RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components or create reusable components. In all cases, automated tools are used to facilitate construction of the software.
  5) **Testing and turnover:** Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

Through above phases it provides the design representations which are basics for RAD.

Like all process models, the RAD approach has drawbacks

1) For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
2) RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.
3) If a system cannot be modularized, building the components necessary for RAD will be problematic.
4) If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
5) RAD is not appropriate when technical risks are high.

2.3 EVOLUTIONARY PROCESS MODELS

Evolutionary models are iterative; they are characterized as follows:

a. Incremental Model:

b. The Spiral Model:

c. The WINWIN Spiral Model

d. The Concurrent Development Model

Incremental model:

- The incremental model combines elements of the waterfall model (applied repetitively) with the iterative fashion.
- Each linear sequence produces a deliverable "increment" of the software
- For example, word-processing software developed using the incremental paradigm will gives in the,

First increment	→ file management, editing,
Second increment	→ more sophisticated editing and document production
Third increment	→ spelling and grammar checking
Fourth increment	→ advanced page layout capability

- When an incremental model is used, the first increment is often a 'core product' (basic requirements). Supplementary features remain undelivered

- The core product is used by the customer. As a result of use, a plan is developed for the next increment

- This process is repeated for each increment, until the complete product is produced

- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project

The Spiral Model:

The spiral is a model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.

The spiral model is divided into a number of framework activities:

1. Customer communication:

   These are required establishing effective communication between developer and customer.

2. Planning:

   These are required to define resources, timelines, and other project related information

3. Risk analysis

   These are required to assess both technical and management risks.

4. Engineering

   These are required to build one or more representations of the application.

5. Construction & release:

   These are required to construct, test, install, and provide user support (e.g., documentation and training).

6. Customer evaluation:

   These are required to obtain customer feedback based on evaluation

- Each region is populated by a series of work tasks these called a '**task set**'.
- Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the objectives, and the constraints that exist.
- In this evolutionary process, the software engineering team moves around the spiral in a clockwise direction, beginning at the center. Each pass through the planning region results in adjustments to the project plan.
- The first circuit around the spiral might result in the development of a product specification.
- Subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- The spiral model is a **realistic approach** to the development of large-scale systems and software.
- It may be difficult to convince customers that the evolutionary approach is controllable.

The WINWIN Spiral Model:

- The objective of this activity is to bring out project requirements from the customer.
- In ideal context the developer simply asks the customer what is required and the customer provides sufficient detail.

- In reality, the customer and the developer enter into a process of negotiation (give and take).

- The best negotiations attempt is a "win-win" result i.e the customer wins by getting product and the developer wins by working to realistic and achievable budgets and goals

The WINWIN spiral model defines the following activities

1. Identification of the system or subsystem's key "**stakeholders**."

2. Determination of the stakeholders' "Win conditions."

3. Negotiation of the stakeholders' Win conditions to reconcile them into a set of win-win conditions

- Successful completion of these initial steps achieves a win-win result.
- In addition to the win-win the model proposes three milestones called anchor points, which are used to complete the cycle. The anchor points are:
  - **Life cycle objectives**: defines set of objectives for each major activity.
  - **Life cycle architecture:** establishes activities that must be met as system and software architecture is defined.
  - **Initial operational capability**: represents objectives associated with software installation.

The Concurrent Development Model:

- The concurrent development model called concurrent engineering. This provides an accurate state of the current state of a project.

- Focus on concurrent engineering activities in a software engineering process such as prototyping, analysis modeling, requirements specification and design.

- Represented schematically as a series of major technical activities, tasks, and their associated states.

- Defined as a series of events that trigger transitions from state to state for each of the software engineering activities.

Two ways to achieve the concurrency:

- System and component activities occur simultaneously and can be modeling using the state-oriented approach

- A typical client/server application is implemented with many components; each can be designed and realized concurrently.

- This is applicable to all types of software development and provides an accurate picture of the current state of a project

2.4 THE UNIFIED PROCESS

- **Unified process** is an attempt to draw on the best features and characteristics of conventional software process models.
- It recognizes the importance of customer communication and makes more efficient methods for describing the customer's view of a system.

Phases of the unified process:

The phases of the unified process are

1. Inception
2. Elaboration
3. construction
4. Transition
5. production

Inception:

- The inception phase of the unified process includes both customer communication and planning activities.
- By working together with the customer and end users, the business requirements for the software are identified, a rough architecture for the system is proposed

Elaboration:

- Elaboration encompasses the customer communication and modeling activities.
- This will refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software they are:
  Use case model, analysis model, design model, implementation model, and deployment model

Construction phase:

- The construction phase develops the software components,
- As components are being implemented, unit tests are designed and executed for each component.

Transition phase:

- Transition phase encompasses the latter stages of construction activity and the first part of the deployment activity.
- At the conclusion of the transition phase the software increment becomes a usable software release.

Production phase:

- It coincides with the deployment activity. In this phase the on going use of the software is monitored,
- Support for the infrastructure is provided, and defect reports and request for changes are submitted and evaluated.

2.5 SOFTWARE REQUIREMENTS

Functional Requirements:

- **Functional requirements** specify which outputs should be produced from the given inputs.
- They describe the relationship between the input and output of the system.
- Functional requirements describe functionality or system services.
- It depends on the type of software, expected users and the type of system where the software is used.
- 'Functional user requirements' may be high-level statements of what the system should do
- But 'functional system requirements' should describe the system services in detail

Here we have a few Examples of functional requirements:

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

Nonfunctional requirements:

- Non functional requirements define system properties and constraints

  E.g. Reliability, response time and storage requirements.
  Constraints are I/O device capability, system representations,

- Process requirements may also be specified permitting a particular CASE system, programming language or development method.

- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

We can classify the non functional requirements, as fallows

Product requirements

- These are the requirements which specify that the delivered product must behave in a particular way
  E.g. execution speed, reliability, etc.

External requirements

- Requirements which arise from factors which are external to the system and its development process

  E.g. Interoperability requirements, legislative requirements, etc.

User requirements:

- User requirements are high-level statements of what the system should do
- User requirements should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.

- User requirements are defined using natural language, tables and diagrams as these can be understood by all users
- But some problems will arise due the usage of natural language, they are as fallows

Lack of clarity:

Precision is difficult without making the document difficult to read.

Requirements confusion:

Functional and non-functional requirements tend to be mixed-up.

Requirements amalgamation:

Several different requirements may be expressed together.

System requirements:

- System requirements are intended to communicate the functions that the system should provide
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.
- System requirements may be defined or illustrated using system models.

Interface specification:

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.
- Three types of interface may have to be defined

1. **Procedural interfaces:**

   Procedural interfaces access to services through calling procedures (API)

2. **Data structures that are exchanged:**

   Data structures are to be passed from one sub-system to another sub system. They can be described with graphical data models.

3. **Data representations.**

   Representations of data will be in bitwise fashion.

- Formal notations are an effective technique for interface specification.

2.6 THE SOFTWARE REQUIREMENT DOCUMENT

- Requirements document states 'what the software will do'. It does not state 'how the software will do it'.
- Requirements document is a *written* statement.
- The main purpose of a requirements document is to serve as an agreement between the developers and the customers on what the application will do.

General Principles in Writing a Requirements Document are:

- Avoid Unnecessary Work by not doing the things that costs more than it worth.
- There should be a detailed document for:

    - large application (many interfaces, screens, lots of features)
    - application with many users or many user workgroups
    - application that is critical to the business

- **Use Iterations:** Iterative approach to writing software is widely accepted but it applies to writing the requirements document.
- **Verify Information:** When collecting requirements it is very important to verify all of the facts by interviewing customers with differing points of view users and management
- **Write to Read**: Write simply and Partition the requirements document into several when necessary.

4.1 DESIGN ENGINEERING

- Software design is an iterative process through which requirements are translated into a "blueprint" for the software
- Software design is the first one among the three technical activities, design, code generation and test;
- Four design models are required for a complete specification of design
    1. Data design
    2. Architectural design,
    3. Interface design
    4. Component design

Data design

- This will transforms the information domain model that was created during analysis into the data structures
- The data objects and relationships and the data dictionary provide the basis for the data design.

Architectural design

- The architectural design defines the relationship between major structural elements of the software

Interface design

- The interface design describes how the software communicates
    - Within itself,
    - With systems that inter operate with it,
    - With humans who use it.
- An interface implies a flow of information, therefore, data and control flow diagrams provide information required for interface design

Component design

- The component-level design transforms structural elements of the software architecture into a procedural description of software components

4.2 DESIGN PROCESS AND DESIGN OUALITY

Through design process the quality of the software can be assessed.

- The fallowing are the characteristics, that a good design process should have,

1. The design must implement all of the explicit requirements containec in the analysis model, and it should contain all of the implicit reauirements desired bv the customer.
2. The design must be a readable, understandable for those who generate code and for those who test the software.
3. The design should provide a complete picture of the software, and functional, and behavioral domains from an implementation point of view

- We can achieve a good quality for design process by using the fallowing guide lines,
    1. A design should be created using recognizable design patterns. It should make up of components that exhibit good design characteristics

2. A design should be modular; that is, the software should be logically partitioned into elements that perform specific functions and sub functions
3. A design should contain distinct representations of data, architecture, interfaces, and components which we call modules
4. A design should lead to data structures that are appropriate for the objects to be implemented
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.
7. A design should be derived using a repeatable method

4.3 DESIGN CONCEPTS

These design concepts will give the foundation from which more sophisticated design methods can be applied.

Abstraction:

- Each step in the software process is a refinement in the level of abstraction of the software solution,
- At different levels of abstraction we will work to create procedural abstraction, data abstraction,

Procedural abstraction:

- Is a named sequence of instructions that has a specific and limited function
- An example of a procedural abstraction would be the word

  'Open' for a door

  Open → walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door.

Data abstraction:

- Data abstraction is a named collection of data that describes a data object
- Example for data abstraction is

Data abstraction for door → door type, swing direction, opening
mechanism, weight, dimensions

- The procedural abstraction 'open' would make use of information contained in the attributes of the data abstraction 'door'

Control abstraction:

- Control abstraction is the third form of abstraction used in software design.
- Like procedural and data abstraction, control abstraction implies a program control mechanism without specifying internal details.
- An example of a control abstraction is the synchronization semaphore used to coordinate activities in an operating system.

Refinement:

- Refinement is actually a process of 'elaboration'.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is equivalent to the process refinement and. The major difference is in the level of implementation detail, instead of the approach
- Abstraction and refinement are complementary concepts. Abstraction enables a designer to specify procedure. Refinement helps the designer to reveal low-level details

Modularity:

- Software is divided into separately named and addressable components, called modules and these modules are integrated to satisfy requirements.
- Modularity is the single attribute of software that allows a program to be intellectually manageable
- Monolithic software i.e. a large program composed of a single module cannot be easily grasped by a reader, to illustrate that consider the fallowing observations

Let,
$C(x) \rightarrow$ complexity of a problem
$E(x) \rightarrow$ effort required to solve problem

If $C(x1) > C(x2)$ it will implies $E(x1)>E(x2)$
Another interesting characteristic is,
$C(x1+x2)>c(x1) + c(x2)$ that implies $E(x1+x2)>E(x1) + E(x2)$

This leads to a "divide and conquer" conclusion
“It's easier to solve a complex problem when you break it into manageable pieces”

- In the above figure the effort to develop an individual software module does decrease as the total number of modules increases,

- However as the number of modules grows, the effort associated with integrating the modules also grows.

Control Hierarchy:

- It also called a program structure represents the organization of program components or modules and implies a hierarchy of control
- Different notations are used to represent control hierarchy

Depth and width: these will provide the indication of the number of levels of control

Fan-out: It is a measure of the number of modules that are directly controlled by another module

Fan-in: It indicates how many modules directly control a given module

Super ordinate: A module that controls another module

Subordinate: A module controlled by another

- Control hierarchy also represents the characteristics of software architecture.

Visibility: this will indicates the set of program components that may be invoked or used as data by a given component

Connectivity: this will indicates the set of components that are directly invoked or used as data by a given component

Structural Partitioning:

- The program structure should be partitioned both horizontally and vertically.

Horizontal partitioning:

- Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.
- The simplest approach to horizontal partitioning defines three partitions

Input,
Data transformation (processing)
Output

- Advantages of horizontal partition:
    - Easy to test, maintain, and extend
    - Fewer side effects in change propagation or error propagation
- Disadvantage:

  More data to passing across module interfaces will complicate the overall control of program flow

Vertical partitioning

- This suggests the control and work should be distributed top down in program structure.

- Advantages:
    - Good at dealing with changes:
    - Easy to maintain the changes
    - Reduce the change impact and propagation

Patterns:

- The design pattern provides a description that enables a designer to determine
    - Whether the pattern is applicable to the current work
    - Whether the pattern can be reused
    - Whether the pattern can serve as guide for developing a similar , but functionally or structurally different pattern

Information hiding:

- Information hiding suggests that the modules should be designed so that information contained within a module is inaccessible to other modules

- Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function
- Major benefits: Reduce the change impacts in testing and maintenance

Functional Independence

- Functional independence is achieved by developing modules, which addresses a specific sub function of requirements

- Software with independent modules, is easier to develop because function may be classified and interfaces are simplified
- Independence is measured using two qualitative criteria: cohesion and coupling.
- 'Cohesion' is a measure of the relative functional strength of a module.
- 'Coupling' is a measure of the relative interdependence among modules.

Refactoring:

- Refactoring is a reorganization technique that simplifies the design of component without changing its function or behavior

- In other words we can say that refactoring is process of changing a software system in such a way that it does not alter the external behavior of the code
- When software is refactored, the existing design is examined for
    - redundancy
    - unused design elements
    - inefficient or unnecessary algorithms
    - poorly constructed or inappropriate data structures

Design Classes:

- There are five different types of design classes; each representing a different layer of the design architecture is suggested.
  User interface classes: these will define all abstractions that are necessary for human computer interaction.

Business domain classes: these will identify the attributes and services that are required to implement some element of the business domain.

Process classes: lower level business abstraction is implemented by these classes.

Persistent classes: These represent the data stores which persist beyond the execution

System classes: these classes implement software management and control functions

4.4 THE DESIGN MODEL

Data elements

Architectural elements

- Application domain
- Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
- Patterns and "styles"

Architectural elements

- Application domain
- Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
- Patterns and "styles"

Interface elements

There are three important elements of interface design,

- The user interface (UI)
- External interfaces to other systems, devices, networks or other producers or consumers of information
- Internal interfaces between various designs components.

Component elements

- The component level design for software describes the internal detail of each software component.

- For this the component level design defines data structures for all local data objects and algorithmic details. r

Deployment elements

- Deployment elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

4.5 CREATING AN ARCHITECTURAL DESIGN

SOFTWARE ARCHITECTURE

- Software architecture refers to "the overall structure of the software and the ways in which that structure provides conceptual integrity for a system"
- It is the hierarchical structure of program components and their interactions

- It is a representation that enables a software engineer to
  1. Analyze the effectiveness of the design in meeting its requirements
  2. Consider architectural alternatives at a stage when making design
     changes is still relatively easy
  3. Reducing the risks associated with the construction of the software.
- Reasons that why software architecture is important

  - For communication between all parties (stakeholders) who interested in the development of a computer based system we need representations of software architecture are an enabler
- The architecture design decisions have a deep impact on all software engineering work

- Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together"

- A set of properties that should be specified as part of an architectural design

Structural properties: The architecture design defines the system components and their interactions.

Extra-functional properties: The architecture design should address how the design architecture achieves requirements for performance, capacity, reliability, adaptability, security.

Families of related systems: The architecture design should draw upon repeatable patterns in the design of families of similar systems

- The architectural design can be represented using different models

Structural models: Represent architecture as an organized collection of program components.

Framework models: Increase the level of design abstraction by attempting to identify repeatable architectural design frameworks

Dynamic models: Address the behavioral aspects of the program architecture

Process models: Focus on the design of the business or technical process that the system must accommodate

Functional models: Functional models can be used to represent the functional hierarchy of a system.

4.6 DATA DESIGN

- Data design translates data objects defined as part of the analysis model into data structures.

Data design at the architectural level:

- Data architecture was generally limited to data structures at the program level and databases at the application level

- In other words the challenges of today's businesses is, maintaining databases serving many applications include the hundreds of gigabytes of data

- To solve this challenge, the business IT community has developed 'data mining' techniques, also called 'knowledge discovery in databases (KDD)'

- The existence of multiple databases, their different structures, and many other factors make data mining difficult

- An alternative is 'data warehouse' which adds additional layers to the data architectures.

- An alternative is 'data warehouse' which adds additional layers to the data architectures.

- A data warehouse is a large, independent database that encompasses some, but not all, of the data that are stored in databases that serve the set of applications required by a business

Data Design at the Component Level

Data design at the component level focuses on the representation of data structures that are directly accessed by one or more software components

The fallowing are the set of principles that may be used to specify and design data structures.

1. The systematic analysis principles which applied to a function and behavior should also be applied to data

2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established and used to define both data and program design

4. Overall data organization may be defined during requirements analysis, refined during data design work, and specified during component level design.

5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure

6. Data structures and operations should be viewed as a resource for software design. Data structures can be designed for reusability.

7. A software design and programming language should support the specification and realization of abstract data types (ADT)

4.7 ARCHITECTURAL STYLES

An architecture style is a transformation that is imposed on the design of an entire system.

- The intention of architecture styles is to establish a structure for all components.

A brief categorization of architectural styles:

Data-centered architectures:

- A data store like a file or database resides at the center of this architecture and is accessed frequently by other components

Data-flow architectures:

- This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.

- A pipe and filter pattern has a set of components, called filters, connected by pipes that transmit data from one component to the next.

- Each filter works independently. They take data input of a certain form, and produces data output of a specified form.

- If the data flow degenerates into a single line of transforms, it is termed batch sequential

Call and return architecture:

This architecture style enables a software designer to achieve a program structure that is relatively easy to modify and scale.

Main program or subprogram architecture:

- This will decomposes function into a control hierarchy. Where a main program invokes a number of program components.

Remote procedure calls architecture:

- The components of main or sub program architecture are distributed across multiple computers on a network.

Object-oriented architectures:

- 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:

- In this architecture number of different layers is defined, as inner layers, outer layers, intermediate layers.
- 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

4.8 ARCHITECTURAL PATTERNS

- An architectural pattern imposes a transform on the design of architecture.

A pattern differs from style in number of ways:

1. The scope of pattern is less broad,
2. A pattern describes how a software will handle some aspects of its functionality

We have different architectural pattern domains,

Concurrency:

- Many applications must handle multiple tasks in a manner that simulates parallelism,

- For example we can consider operating system management pattern, which allows execution of components concurrently.

Persistence:

- Data persists if it survives past the execution of the process that created it.
- Persistent data are stored in a database of file and may be read or modified by other processes at later time.
- For example word processing software that manages its own document structure.

Distribution:

- The distribution problem addresses the manner in which system or components within the systems communicate with one another in distributed environment
- There are two elements to this problem

  1. The way in which entities connect to one another
  2. The nature of the communication that occurs.

- The pattern that address this problem is broker pattern
- The broker acts as middle man, between client and server, the client sends the message to the broker, broker will complete the connection.

4.9 ARCHITECTURAL DESIGN

- The architecture design begins with representing the system in context i.e. defining the external entities,
- Once all external software interfaces have been described, the designer specifies the structure of the system by defining and refining software components.

Representing the system in context:

The software architect uses an architectural context diagram (ACD), to model the manner in which software interacts with entities to its boundaries.

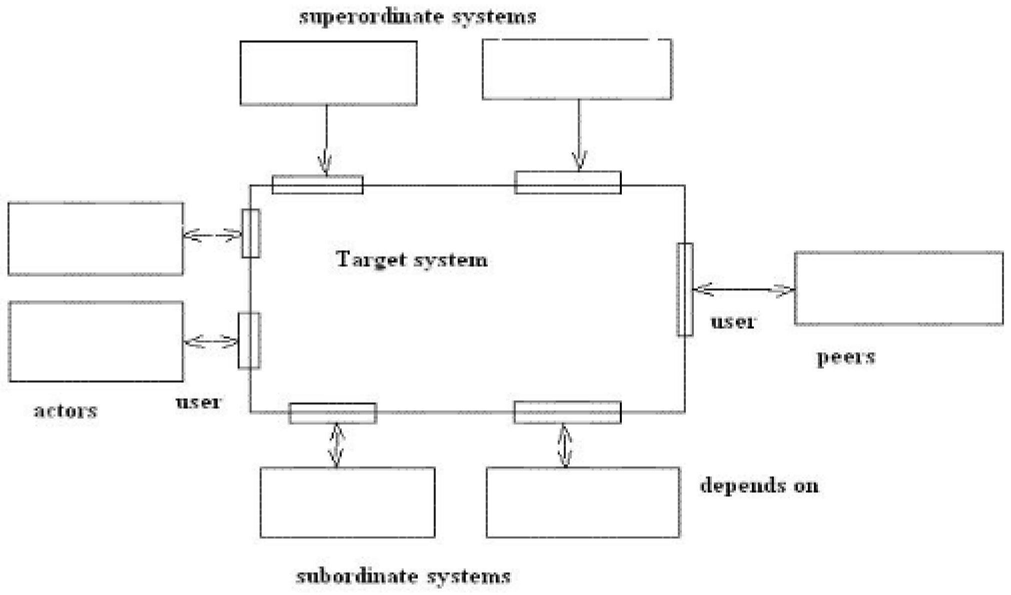
Super ordinate systems: These systems use the target system as part of some higher level processing scheme

Subordinate Systems: These systems that are used by the target system provide data or processing that is necessary to complete target system functionality.

Peer level systems: these systems that interact on a peer to peer basis. That is information is either produced or consumed by the peers and target

Actors: these entities that interact with the target system by producing or consuming information that is necessary for requisite processing.

Each of these entities that interact with the target systems trough an interface



The architecture context diagram

Defining Archetypes:

- An 'archetype' is a class or pattern that represents a core abstraction which is critical to the design of architecture for target system.

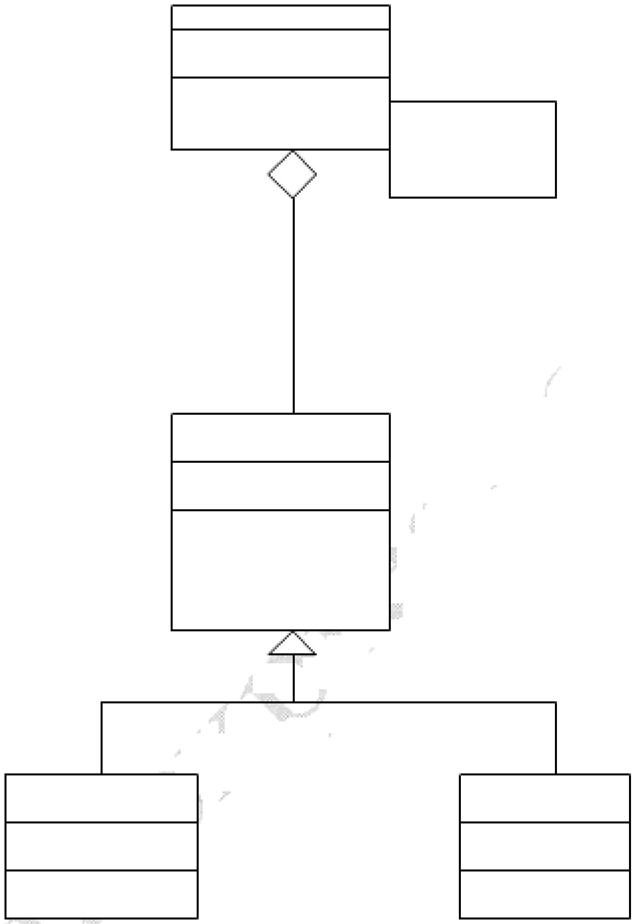
- For example we can define the fallowing archetypes for a 'safe home security' :

Node: Represents the interconnected input output components.

Detector: It encompasses the logic that provides information into the target system.

Indicator: Represents all mechanisms (like alarm siren, flash light, bells), for indicating that an alarm condition is occurring.

Controller: This shows the mechanism that allows the supporting non supporting node. If controller resides on a network, they have the ability to communicate with one another.



Refining the architecture into components:

- As the architecture is refined into components, the structure of the system begins to come out.
- The application domain is one source for the derivation and refinement of components.
- Another source is infrastructure domain
- Example, the memory management components, communication components, database components, and task management components are often integrated into the software architecture.

Describing instantiation of the system:

- The architectural design that has been modeled to this point is still relatively high level.
- At this level, the overall structure of the system is apparent, and major software components have been identified.
- In the final stage the instantiation of the architecture is developed.