Course Outcomes:

After completing the course student should be able to:

1. Describe the Fundamentals of software architecture, qualities and terminologies.
2. Understand the fundamental principles and guidelines for software architecture design, architectural styles, patterns, and frameworks.
3. Use implementation techniques of Software architecture for effective software development.
4. Apply core values and principles of software architectures for enterprise application development.

Course Contents:

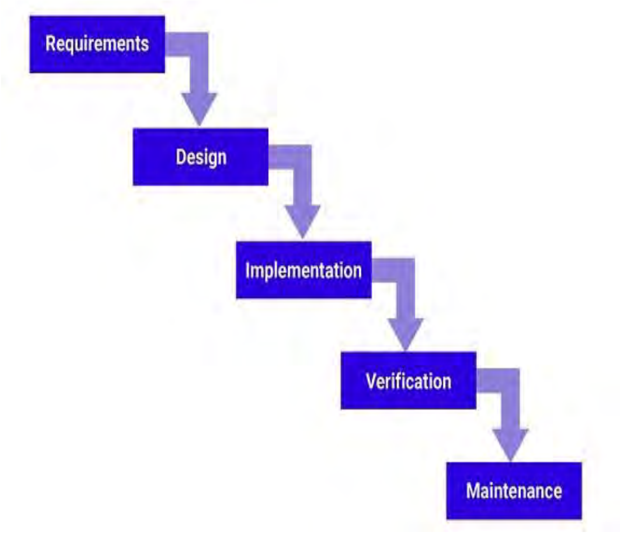
Unit 1. Overview of Software development methodology and software quality model, different models of software development and their issues. Introduction to software architecture, evolution of software architecture, software components and connectors, common software architecture frameworks, Architecture business cycle – architectural patterns – reference model.

Overview of Software Development Methodology:

Software development methodology is a framework that is used to structure, plan, and control the process of developing an information system. This kind of development methodologies are only concerned with the software development process, so it does not involve any technical aspect of, but only concern with proper planning for the software development.

There are Various Software development Methodology:

1. Waterfall Development Model:



The waterfall model is one of the most traditional and commonly used software development methodologies for software development. This life cycle model is often considered as the classic style of the software development. This model clarifies the software development process in a linear sequential flow that means that any phase in the development process begins only if the earlier phase is completed. This development approach does not define the process to go back to the previous phase to handle changes in requirements.

Figure 1.1 Waterfall Model

Advantages of Waterfall Model:

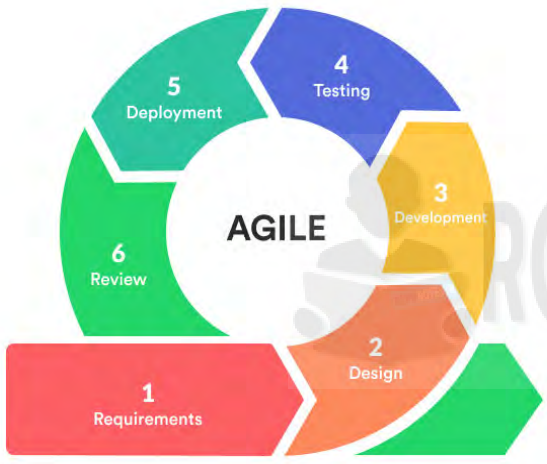
1. Waterfall model is very simple and easy to understand and use a method that is why it is really beneficial for the beginner or novice developer.

2. It is easy to manage, because of the rigidity of the model. Moreover, each phase has specific deliverables and individual review process.
3. In this model phases are processed and completed are at once in a time thus it saves a significant amount of time.
4. This type of development model works more effectively in the smaller projects where requirements are very well understood.
5. The testing is easier as it can be done by reference to the scenarios defined in the earlier functional specification.

Disadvantages of Waterfall Model:

1. This model can only be used when very precise up-front requirements are available.
2. This model is not applicable for maintenance type of projects.
3. The main drawback of this method is that once an application is in the testing stage, it is not possible to go back and edit something.
4. There is no possibility to produce any working software until it reaches the last stage of the cycle.
5. In this model, there is no option to know the end result of the entire project.

2. Agile Software Development:



Agile Software Development is an approach that is used to design a disciplined software management process which also allows some frequent alteration in the development project. This is a type of software development methodologies which is one conceptual framework for undertaking various software engineering projects. It is used to minimize risk by developing software in short time boxes which are called iterations that generally last for one week to one month.

Figure 1.2 Agile Software Development

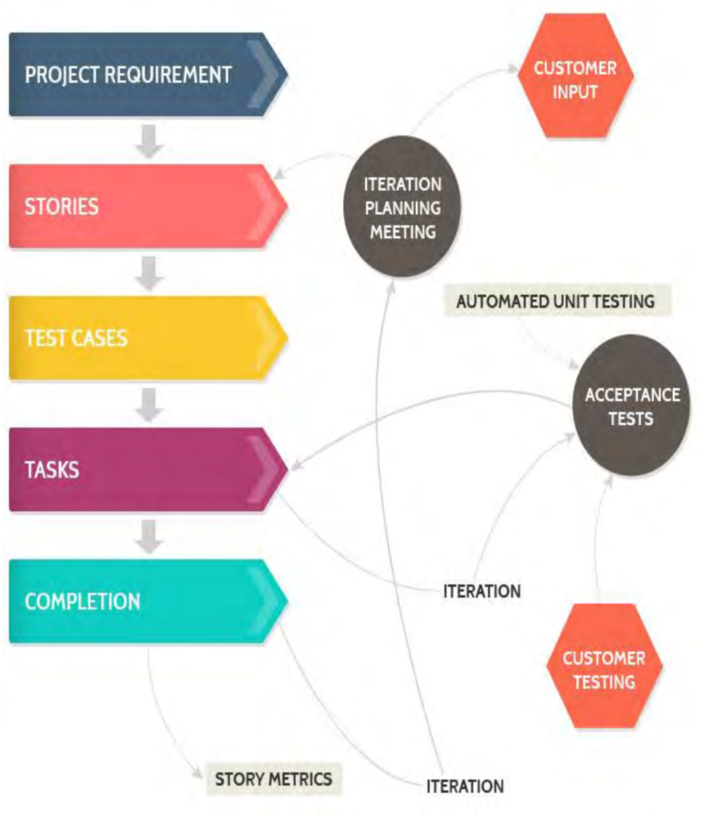
Advantages of Agile Development Methodology:

1. Agile methodology has an adaptive approach which is able to respond to the changing requirements of the clients.
2. Direct communication and constant feedback from customer representative leave no space for any guesswork in the system.

Disadvantage of Agile Development Methodology:

1. This methodology focuses on working software rather than documentation, hence it may result in a lack of documentation.
2. The software development project can get off track if the customer is not very clear about the outcome of his project.

3. Extreme Programming: Extreme Programming is an agile software engineering methodology. This methodology, which is shortly known as XP methodology is mainly used for creating software within a very unstable environment. It allows greater flexibility within the modelling process. The main goal of this XP model is to lower the cost of software requirements. It is quite common in the XP model that the cost of changing the requirements on later stage in the project can be very high.



Advantages of Extreme Programming:

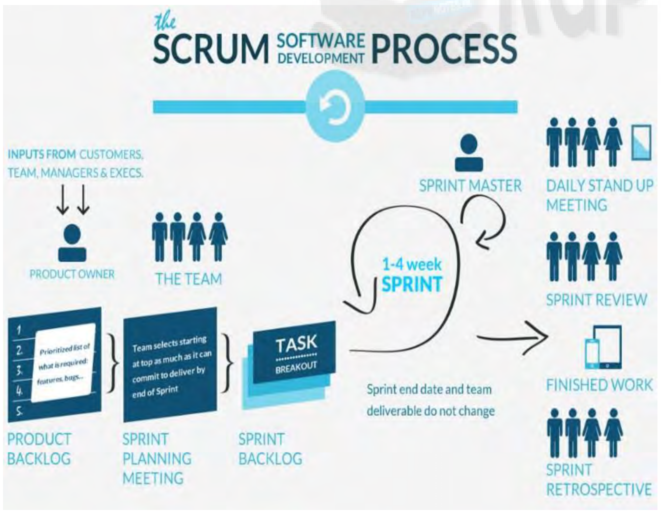
1. It emphasis on customer involvement.
2. This model helps to establish rational plans and schedules and to get the developers personally committed to their schedules which are surely a big advantage in the XP model.
3. This model is consistent with most modern development methods so, developers are able to produce quality software.

Disadvantages of Extreme Programming:

1. It requires meetings at frequent intervals at enormous expense to customers.
2. It requires too much development changes which are really very difficult to adopt every time for the software developer.
3. It tends to impossible to be known exact estimates of work effort needed to provide a quote, because at the starting of the project nobody aware about the entire scope and requirements of the project

Figure 1.3 Extreme Programming Software Development Methodology

4. Scrum Development Methodology:


the SCRUM SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
Sprint end date and team deliverable do not change
FINISHED WORK
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

The Scrum Development Methodology can be applied to nearly any project. This process is suited for development projects that are rapidly changing or highly emergent requirements. The Scrum software development model begins with a brief planning, meeting and concludes with a final review. This development methodology is used for speedy development of software which includes a series of iterations to create required software. It is an ideal methodology because it easily brings on track even the slowest progressing projects.

Figure 1.4 Scrum Software Development Methodology

Advantages of Scrum Development:

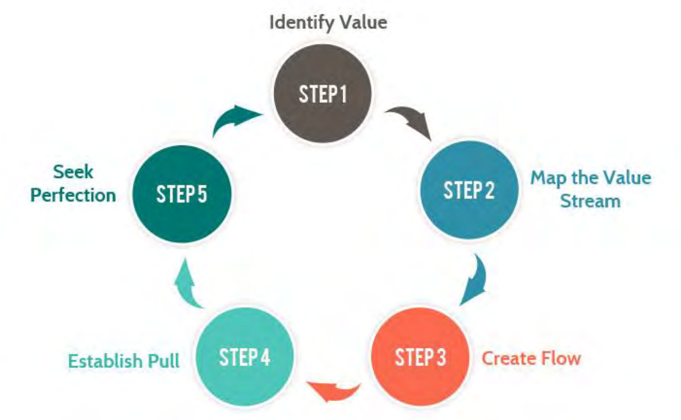
1. In this methodology, decision-making is entirely in the hands of the teams.
2. This methodology enables projects where the business requirements documentation is not considered very significant for the successful development.
3. It is a lightly controlled method which totally empathizes on frequent updating of the progress, therefore, project development steps are visible in this method.

Disadvantages of Scrum Development:

1. This kind of development model is suffered if the estimating project costs and time will not be accurate.
2. It is good for small, fast moving projects but not suitable for large size projects.

3. This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed within exact time frame.

5. Lean Development Methodology:



Lean Development Methodology focuses on the creation of easily changeable software. This Software Development model is more strategically focused than any other type of agile methodology. The goal of this methodology is to develop software in one-third of the time, with very limited budget, and very less amount of required workflow.

Figure 1.5 Lean Software Development Methodology

Advantages of Lean Development Methodology:

1. The early elimination of the overall efficiency of the development process certainly helps to speeds up the process of entire software development which surely reduces the cost of the project.
2. Delivering the product early is a definite advantage. It means that development team can deliver more functionality in a shorter period of time, hence enabling more projects to be delivered.
3. Empowerment of the development team helps in developing the decision-making ability of the team members which created more motivation among team members.

Disadvantages of Lean Development Methodology:

1. Success in the software development depends on how disciplined the team members are and how to advance their technical skills.
2. The role of a business analyst is vital to ensure the business requirements documentation is understood properly. If any organization doesn't have a person with the right business analyst, then this method may not be useful for them.
3. In this development model, great flexibility is given to developer, which is surely great, but too much of it will quickly lead to a development team who lost focus on its original objectives thus, it hearts the flow of entire project development work.

6. Rapid Application Development Methodology: Rapid Application Development (RAD) is an effective methodology to provide much quicker development and higher-quality results than those achieved with the other software development methodologies. It is designed in such a way that, it easily take the maximum advantages of the software development. The main objective of this methodology is to accelerate the entire software development process. The goal is easily achievable because it allows active user participation in the development process.

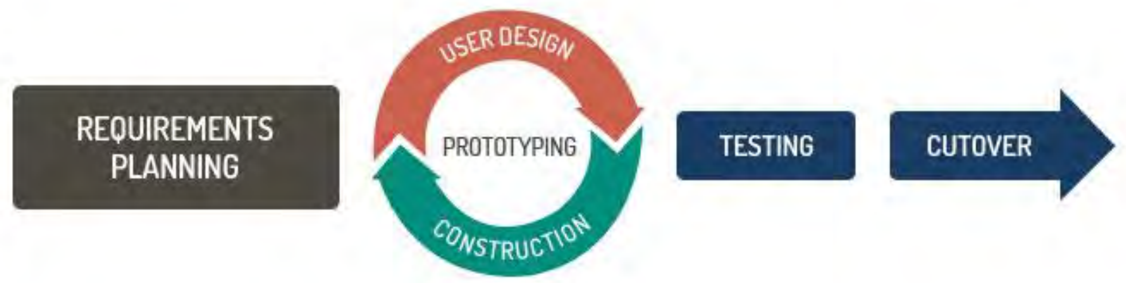


Figure 1.6 Rapid Application Development Methodology

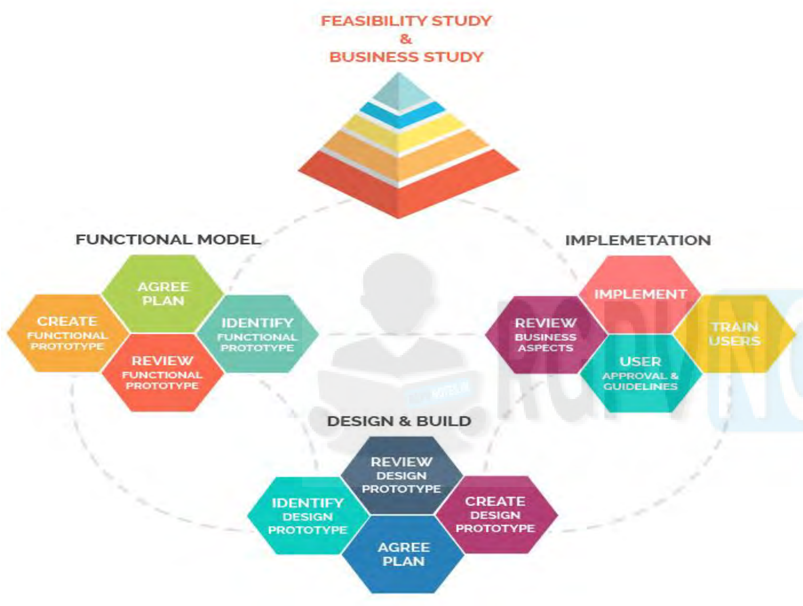
Advantages of the RAD model:

1. Rapid Application development model helps to reduce the risk and required efforts on the part of the software developer.
2. This model also helps clients to take quick reviews for the project.
3. This methodology encourages customer feedback which always provides improvement scope for any software development project.

Disadvantages RAD model:

1. This model depends on the strong team and individual performances for clearly identifying the exact requirement of the business.
2. This approach demands highly skilled developers and designer's team which may not be possible for every organization.
3. This method is not applicable for the developer to use in small budget projects as a cost of modelling and automated code generation is very high.

7. Dynamic Systems Development Model Methodology:


FEASIBILITY STUDY & BUSINESS STUDY
FUNCTIONAL MODEL
AGREE PLAN
CREATE FUNCTIONAL PROTOTYPE
IDENTIFY FUNCTIONAL PROTOTYPE
REVIEW FUNCTIONAL PROTOTYPE
IMPLEMETATION
IMPLEMENT
REVIEW BUSINESS ASPECTS
TRAIN USERS
USER APPROVAL & GUIDELINES
DESIGN & BUILD
REVIEW DESIGN PROTOTYPE
IDENTIFY DESIGN PROTOTYPE
CREATE DESIGN PROTOTYPE
AGREE PLAN

Dynamic Systems Development Model is a software development methodology originally based on the Rapid Application Development methodology. This is an iterative and incremental approach that emphasizes continuous user involvement. Its main aim is to deliver software systems on time and on the budget. This model simply works on the philosophy that nothing is developed perfectly in the first attempt and considers as an ever-changing process.

Figure 1.7 Dynamic System Development Methodology

Advantages of Dynamic Systems Development Model:

1. Users are highly involved in the development of the system so, they are more likely to get a grip on the software development project.
2. In this model, basic functionality is delivered quickly, with more functionality being delivered at frequent intervals.

Disadvantages of Dynamic Systems Development Model:

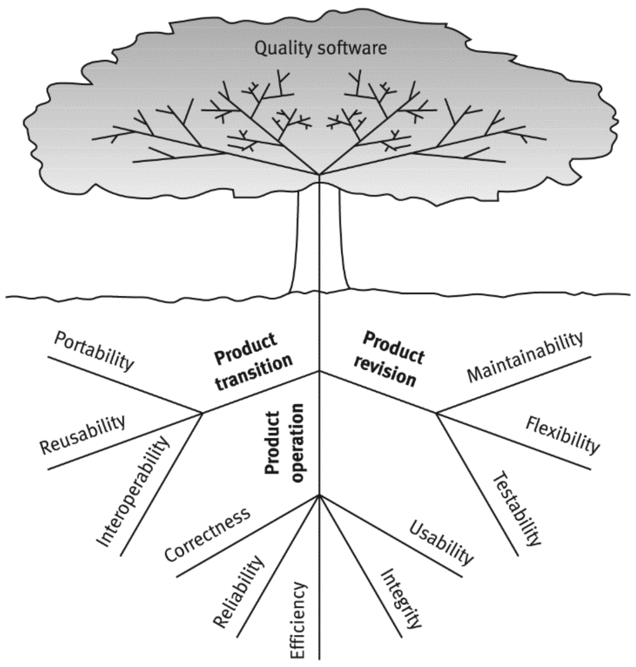
1. The first thing is DSDM is costly to implement, as it requires users and developers both to be trained to employ it effectively. It may not be suitable for small organizations or one-time projects.
2. It is a relatively new model, therefore, it is not very common and easy to understand.

Software Quality Model

Software Quality Models are a standardised way of measuring a software product. With the increasing trend in software industry, new applications are planned and developed every day. This eventually gives rise to the need for reassuring that the product so built meets at least the expected standards.

Following are few models that explains what kind of quality criteria is to be followed.

1. **Mc Call's Model -**



It is the first quality model developed, which defines a layout of the various aspects that define the product's quality. It defines the product quality in the following manner – Product Revision, Product Operation, Product Transition. Product revision deals with maintainability, flexibility and testability, product operation is about correctness, reliability, efficiency and integrity.

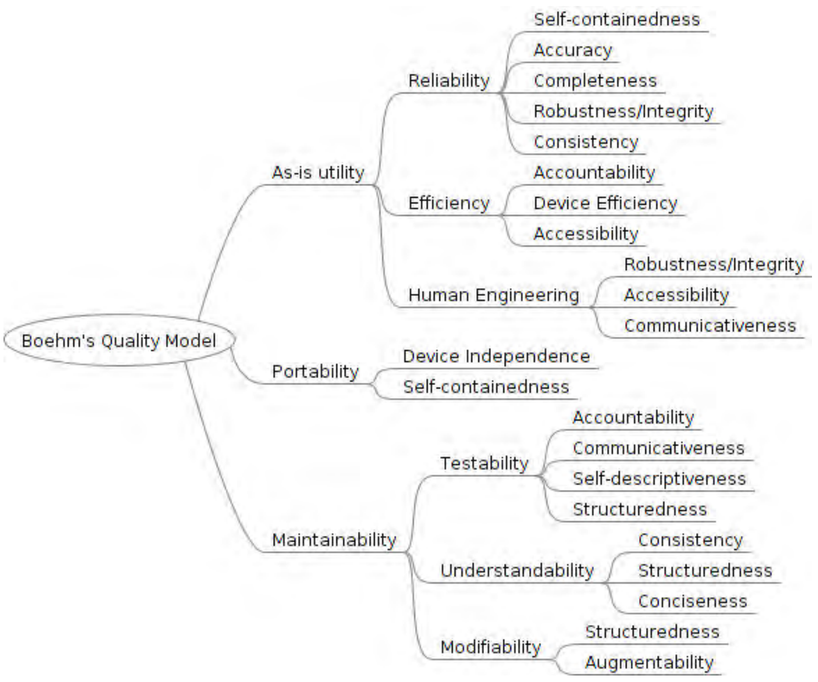
(i) Product Revision - The product revision perspective identifies quality factors that influence the ability to change the software product, these factors are:-**(a) Maintainability -** The ability to find and fix a defect. **(b) Flexibility -** The ability to make changes required as dictated by the business. **(c) Testability-** The ability to Validate the software requirements.

Figure 1.8 Mc Call's Model

(ii) Product Transition -The product transition perspective identifies quality factors that influence the ability to adapt the software to new environments: - **(a) Portability** -The ability to transfer the software from one environment to another. **(b) Reusability** - The ease of using existing software components in a different context. **(c) Interoperability** - The extent, or ease, to which software components work together.

(iii) Product Operations - The product operations perspective identifies quality factors that influence the extent to which the software fulfils its specification -**(a)Correctness** - The functionality matches the specification.**(b)Reliability** - The extent to which the system fails.**(c) Efficiency -** System resource (including CPU, disk, memory, network) usage.**(d) Integrity** - Protection from unauthorized access.**(e) Usability -**ease of use.

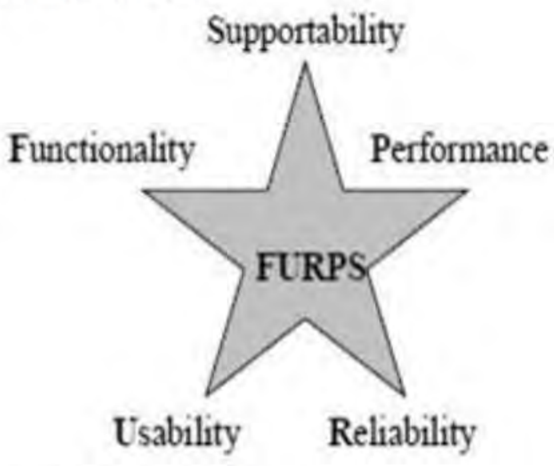
2. Boehm Model -



It describes how easily and reliably a software product can be used. This model actually elaborates the aspects of McCall model in detail. It begins with the characteristics that resorts to higher level requirements. The model's general utility is divided into various factors - portability, efficiency and human engineering, which are the refinement of factors like portability and utility. Further maintainability is refined into testability, understand ability and modifiability.

Figure 1.9 Bohem Model

3. FURPS Model -



This model categorises requirements into functional and non-functional requirements. The term FURPS is an acronym for Functional requirement(F) which relies on expected input and output, and in non-functional requirements (U) stands for Usability which includes human factors, aesthetic, documentation of user material of training, (R) stands for reliability(frequency and severity of failure, time among failure), (P) stands for Performance that includes functional requirements, and finally (S) stands for supportability that includes backup, requirement of design and implementation etc.

Figure 1.10 FURPS Model

FURPS is a checklist for requirements, which help maintain a SQ Standard. It compromises:- (a) Functional (features, capabilities, security), (b) Usability (human factors, help, documentation) (c) Reliability (frequency of failure, recoverability, predictability), (d) Performance (response time, throughput, accuracy, availability, resource usage), (e) Supportability (adaptability, maintainability, internationalization, configurability)

4. ISO 9126 Quality Model -

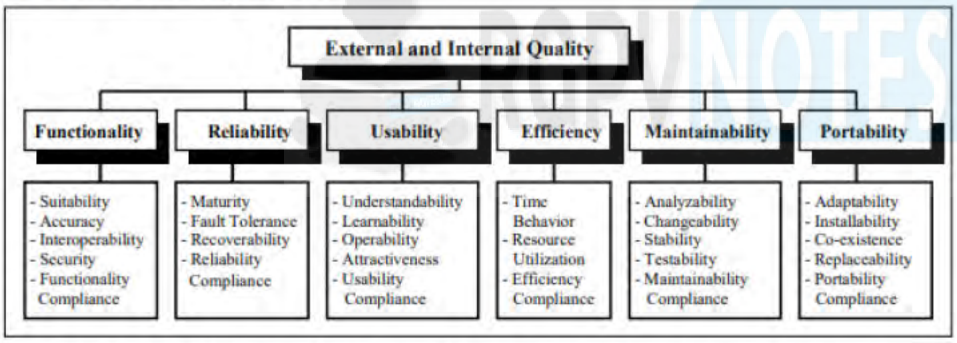


Figure 1.11 ISO 9126 Quality Model for External and Internal Quality

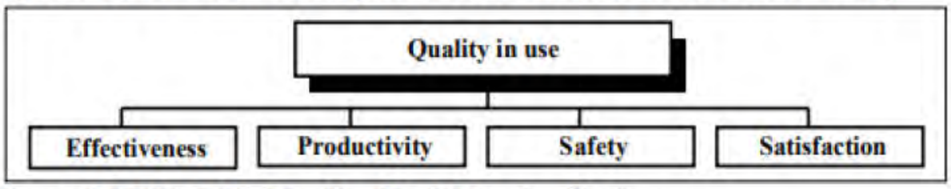


Figure 1.12 ISO 9126 Quality Model for Quality in use

It has two primary categories – internal and external quality attributes and quality in use attributes. The internal quality attributes are the properties of the system the evaluation of which can be done without executing it. Whereas the external quality attributes are those that are evaluated by observing the system during execution.

Different Models of Software Development and their issues

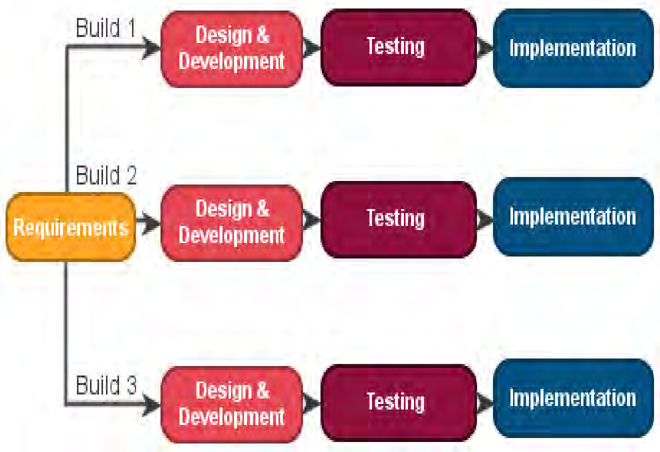
1. **Waterfall Model** - The waterfall model is one of the most traditional and commonly used software development methodologies for software development. This life cycle model is often considered as the classic style of the software development. This model clarifies the software development process in a linear sequential flow that means that any phase in the development process begins only if the earlier phase is completed.

Issues

- This model can only be used when very precise up-front requirements are available.
- The main issue of this method is that once an application is in the testing stage, it is not possible to go back and edit something.

- There is no possibility to produce any working software until it reaches the last stage of the cycle.
- In this model, there is no option to know the end result of the entire project.
- This model is good for a small project but not ideally suitable for long and ongoing projects.

2. Iteration Model-



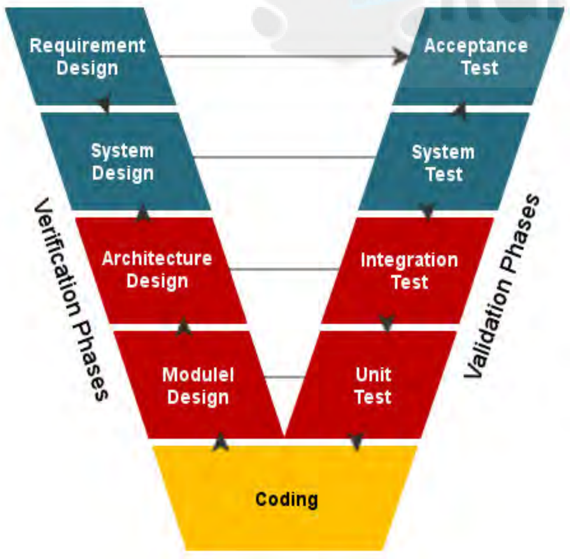
An iterative life cycle model does not start with a full specification of requirements. In this model, the development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. Each release of Iterative Model is developed in a specific and fixed time period, which is called iteration. Furthermore, this iteration focuses on a certain set of requirements. Each cycle ends with a usable system i.e., a particular iteration results in an executable release.

Figure 1.13 Iterative Model

Issues

- More resources may be required, and more management attention is required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- It is not suitable for smaller projects, and highly skilled resources are required for skill analysis.
- Project progress is highly dependent upon the risk analysis phase.
- Defining increments may require definition of the complete system.

3. V-Shaped Model-



V-Model is an SDLC model that has a testing phase corresponding to every development stage in the waterfall model. It is pronounced as the "vee" model. The V-model is an extension of the waterfall model. V model Testing is done in parallel to development. It is also called a Validation and Verification Model.

Issues

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

Figure 1.14 V- Model

4. **Rapid Application Development Methodology:** Rapid Application Development (RAD) is an effective methodology to provide much quicker development and higher-quality results than those achieved with the other software development methodologies. It is designed in such a way that, it easily takes the maximum advantages of the software development. The main objective of this methodology is to accelerate the entire software development process. The goal is easily achievable because it allows active user participation in the development process.

Issues

- This model depends on the strong team and individual performances for clearly identifying the exact requirement of the business.

- It only works on systems that can be modularized can be built using this methodology.
- This approach demands highly skilled developers and designer's team which may not be possible for every organization.
- This method is not applicable for the developer to use in small budget projects as a cost of modelling and automated code generation is very high.

5. Spiral Model-

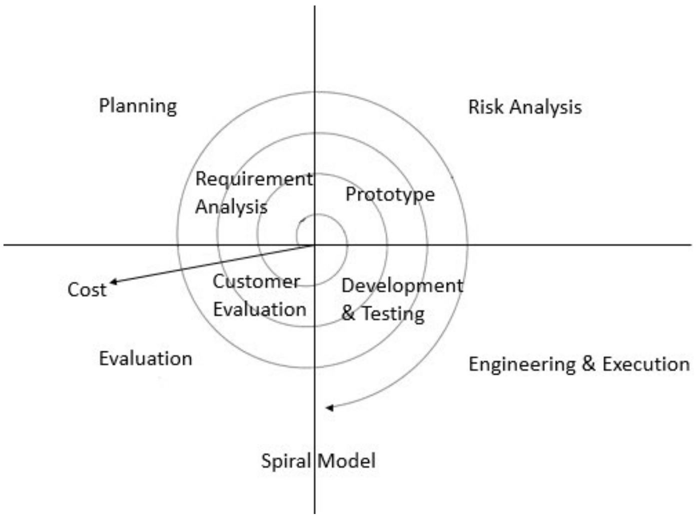


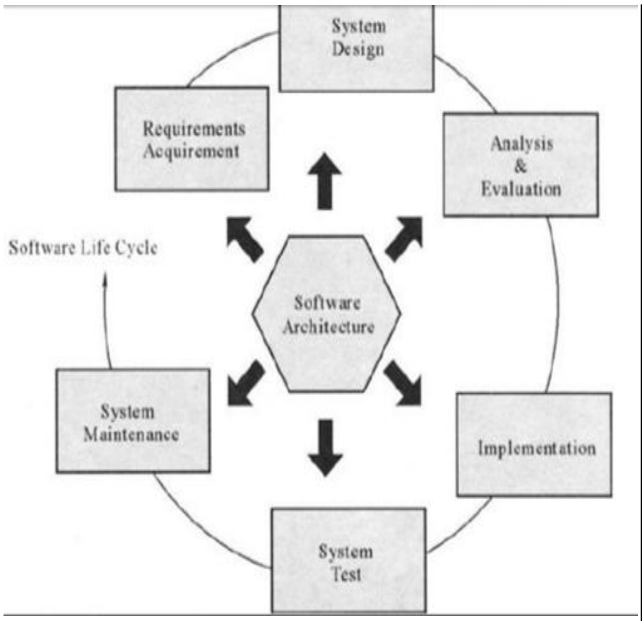
Figure 1.15 Spiral Model

Spiral Model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper. The development team in Spiral-SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

Issues

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Software Architecture



"The software architecture of a program or computing system is the structure or structures of the system, which comprise- Software components, The externally visible properties of those components, The relationships among them. "The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time." It encompasses the set of significant decisions about the organization of a software system.

- Selection of the structural elements and their interfaces by which a system is composed.
- Behaviour as specified in collaborations among those elements.

Figure 1.16 Software Architecture

There are fundamentally three reasons for software architecture's importance from a technical perspective.

- Communication among stakeholders
- Early design decisions
- Transferable abstraction of a system

Evolution of Software Architecture

Software architecture has been an evolutionary discipline starting with monolithic mainframes to recent micro services.

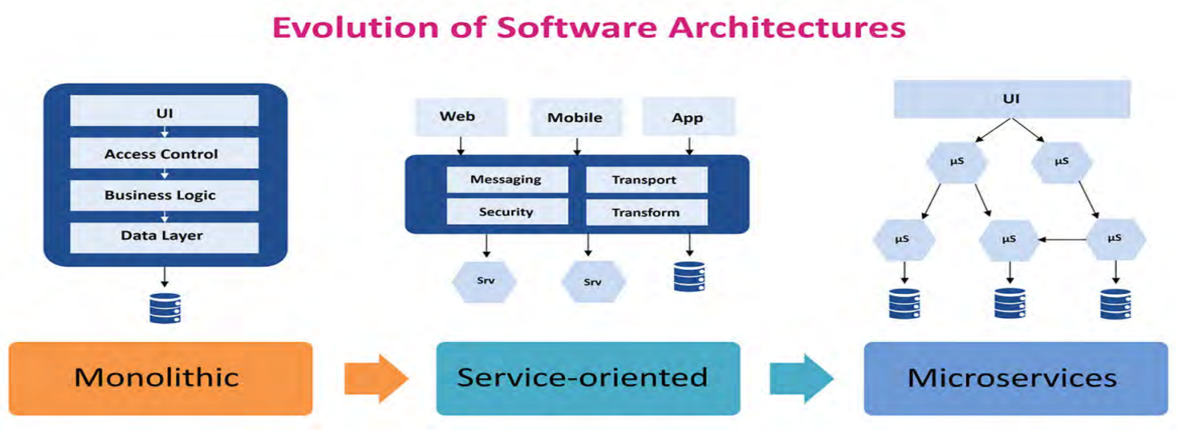


Figure 1.17 Evolution of Software Architecture

The mainframe era was of expensive hardware, having powerful server capable of processing large amount instructions and client connecting to it via dumb terminals. This evolved as hardware became cheaper and dumb terminals paved way to smart terminals. These smart terminals had reasonable processing power leading to a client server models. There are many variations of client server models around how much processing a client should do versus the server. For instance, client could all the processing having server just act as centralized data repository. The primary challenge with that approach was maintenance and pushing client-side updates to all the users. This led to using browser clients where the UI is essentially rendered from server in response to a HTTP request from browser.

As server started having multiple responsibilities in this new world like serving UI, processing transactions, storing data and others, architects broke the complexity by grouping these responsibilities into logical layers – UI Layer, Business Layer, Data Layer, etc. Specific products emerged to support these layers like Web Servers, Database servers, etc. Depending on the complexity these layers were physically separated into tier. The word tier indicates a physical separation where Web Server, Database server and business processing components run on their own machines.

Software Component and Connectors

Component - A Component is a unit of behaviour. Its description defines what the component can do and what it requires to do that job.

Connector - A Connector is an indication that there is a mechanism that relates one component to another usually through relationships such as data flow or control flow.

Changing the grouping of behaviours in components or changing which components are connected changes the value of certain quality attributes.

Component-and-connector structures help answer questions such as-

- What are the major executing components and how do they interact?
- What are the major shared data stores?
- Which parts of the system are replicated? How does data progress through the system?
- What parts of the system can run in parallel?
- How can the system's structure change as it executes?

Process, or communicating processes - Like all component-and-connector structures, this one is orthogonal to the module-based structures and deals with the dynamic aspects of a running system. The units here are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations. The relation in this (and in all component-and-connector structures) is *attachment*, showing how the components and connectors are hooked together. The process structure is important in helping to engineer a system's execution performance and availability.

Concurrency - This component-and-connector structure allows the architect to determine opportunities for parallelism and the locations where resource contention may occur. The units are components and the connectors are "logical threads." A logical thread is a sequence of computation that can be allocated to a separate physical thread later in the design process. The concurrency structure is used early in design to identify the requirements for managing the issues associated with concurrent execution.

Shared data, or repository - This structure comprises components and connectors that create, store, and access persistent data. If the system is in fact structured around one or more shared data repositories, this structure is a good one to illuminate. It shows how data is produced and consumed by runtime software elements, and it can be used to ensure good performance and data integrity.

Client-server - If the system is built as a group of cooperating clients and servers, this is a good component-and-connector structure to illuminate. The components are the clients and servers, and the connectors are protocols and messages they share to carry out the system's work. This is useful for separation of concerns (supporting modifiability), for physical distribution, and for load balancing (supporting runtime performance).

Common Software Architecture Framework

The software needs the architectural design to represent the design of software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectural styles. Each style will describe a system category that consists of:

- A set of components (e.g.: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.

1. **Layered Architecture:** This pattern can be used to structure programs that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer. The most commonly found 4 layers of a general information system are as follows.

  - Presentation layer (also known as UI layer)
  - Application layer (also known as service layer)
  - Business logic layer (also known as domain layer)
  - Data access layer (also known as persistence layer)

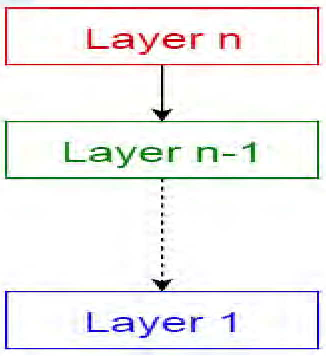


Figure 1.18 Layered Architecture

Usage

1. General Desktop application.
2. E- commerce application.

2. Client Server Architecture - This pattern consists of two parties; a server and multiple clients. The server component will provide services to multiple client components. Clients request services from the server and the server provides relevant services to those clients. Furthermore, the server continues to listen to client requests.

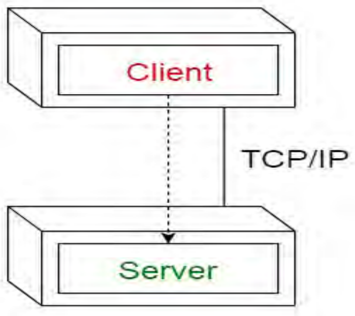


Figure 1.19 Client Server Architecture

Usage

1. Online applications such as email, document sharing and banking.

3. **Master Slave** - This pattern consists of two parties; master and slaves. The master component distributes the work among identical slave components and computes a final result from the results which the slaves return.

Usage

- In database replication, the master database is regarded as the authoritative source, and the slave databases are synchronized to it.
- Peripherals connected to a bus in a computer system (master and slave drives).

4. **Model View Controller-**

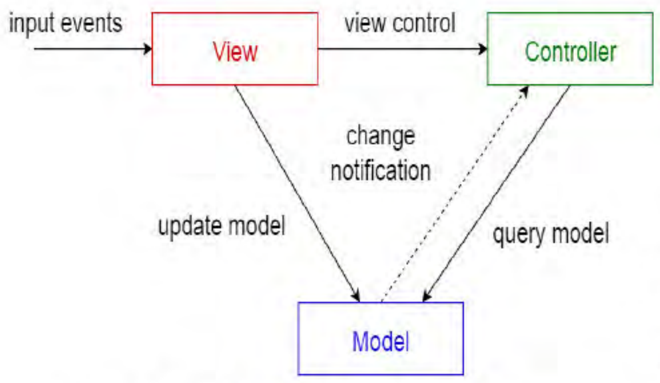


Figure 1.20 Model View Controllers

This pattern, also known as MVC pattern, divides an interactive application in to 3 parts as,

model — contains the core functionality and data
view — displays the information to the user (more than one view may be defined)
controller — handles the input from the user

This is done to separate internal representations of information from the ways information is presented to, and accepted from, the user. It decouples components and allows efficient code reuse.

Usage

- Architecture for World Wide Web applications in major programming languages.
- Web frameworks such as Django and Rails.

Architecture Business Cycle

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software architecture is a result of technical, business and social influences. Its existence in turn affects the technical, business and social environments that subsequently influence future architectures. We call this cycle of influences, from environment to the architecture and back to the environment, the **Architecture Business Cycle (ABC).**

Working of architecture business cycle:

1) The architecture affects the structure of the developing organization. An architecture prescribes a structure for a system it particularly prescribes the units of software that must be implemented and integrated to form the system. Teams are formed for individual software units; and the development, test, and integration activities around the units. Likewise, schedules and budgets allocate resources in chunks corresponding to the units. Teams become embedded in the organization's structure. This is feedback from the architecture to the developing organization.

2) The architecture can affect the goals of the developing organization. A successful system built from it can enable a company to establish a foothold in a particular market area. The architecture can provide opportunities for the efficient production and deployment of the similar systems, and the organization may adjust its goals to take advantage of its newfound expertise to plumb the market. This is feedback from the system to the developing organization and the systems it builds.

3) The architecture can affect customer requirements for the next system by giving the customer the opportunity to receive a system in a more reliable, timely and economical manner than if the subsequent system were to be built from scratch.

4) The process of system building will affect the architect's experience with subsequent systems by adding to the corporate experience base.

An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used.
For ex: client-server is a common architectural pattern. Client and server are two element types, and their coordination is described in terms of the protocol that the server uses to communicate with each of its clients.
A reference model is a division of functionality together with data flow between the pieces.
A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem.
A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. Whereas a reference model divides the functionality, a reference architecture is the mapping of that functionality onto a system decomposition.

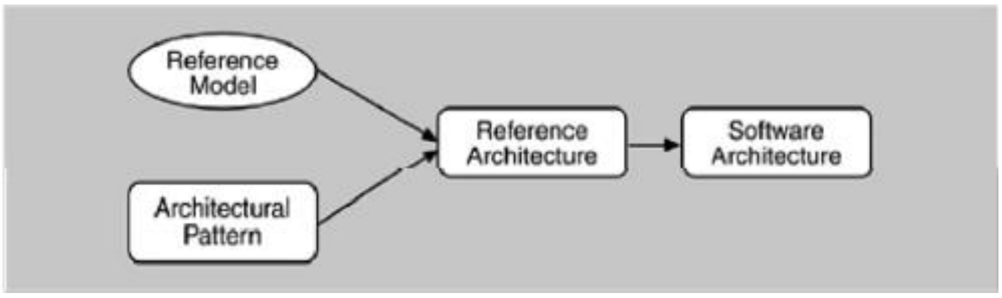


Figure 1.21 Relationship of Reference models, architectural patterns, and reference architectures

Reference models, architectural patterns, and reference architectures are not architectures; they are useful concepts that capture elements of an architecture. Each is the outcome of early design decisions. A software architect must design a system that provides concurrency, portability, modifiability, usability, security, and the like, and that reflects consideration of the trade-offs among these needs.