Kalinga University

Faculty of Computer Science & IT

Course- BCAAIML
Subject- Software Engineering and Testing
Subject Code –BCAAIML503 **Sem- 5th**

Unit I

Software Evolution - Software Engineering

The software evolution process includes fundamental activities of change analysis, release planning, system implementation, and releasing a system to customers.

1. The cost and impact of these changes are accessed to see how much the system is affected by the change and how much it might cost to implement the change.
2. If the proposed changes are accepted, a new release of the software system is planned.
3. During release planning, all the proposed changes (fault repair, adaptation, and new functionality) are considered.
4. A design is then made on which changes to implement in the next version of the system.
5. The process of change implementation is an iteration of the development process where the revisions to the system are designed, implemented, and tested.

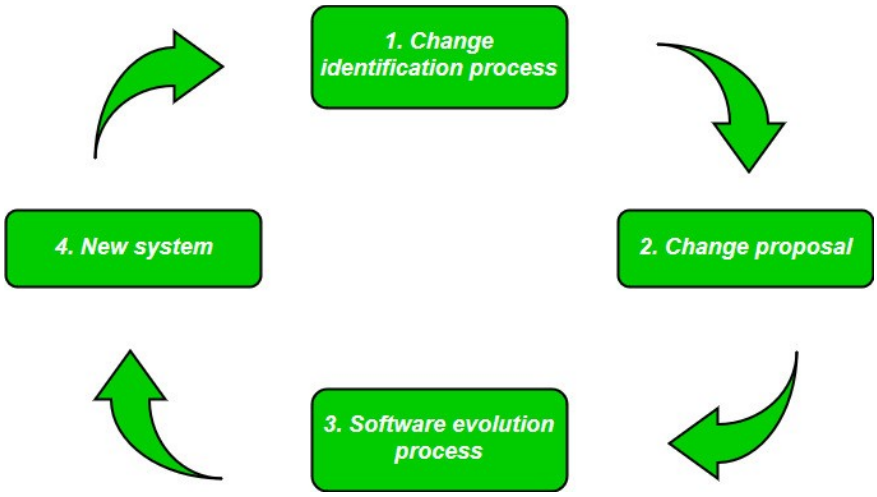
Necessity of Software Evolution

Software evaluation is necessary just because of the following reasons:

1. **Change in requirement with time:** With time, the organization's needs and modus Operandi of working could substantially be changed so in this frequently changing time the tools(software) that they are using need to change to maximize the performance.
2. **Environment change:** As the working environment changes the things(tools) that enable us to work in that environment also changes proportionally same happens in the software world as the working environment changes then, the organizations require reintroduction of old software with updated features and functionality to adapt the new environment.
3. **Errors and bugs:** As the age of the deployed software within an organization increases their preciseness or impeccability decrease and the efficiency to bear the increasing complexity workload also continually degrades. So, in that case, it becomes necessary to avoid use of obsolete and aged software. All such obsolete Pieces of software need to undergo the evolution process in order to become robust as per the workload complexity of the current environment.
4. **Security risks:** Using outdated software within an organization may lead you to at the verge of various software-based cyberattacks and could expose your confidential data illegally associated with the software that is in use. So, it becomes necessary to avoid such security

breaches through regular assessment of the security patches/modules are used within the software. If the software isn't robust enough to bear the current occurring Cyber attacks so it must be changed (updated).

5. **For having new functionality and features:** In order to increase the performance and fast data processing and other functionalities, an organization need to continuously evolute the software throughout its life cycle so that stakeholders & clients of the product could work efficiently.



Laws used for Software Evolution

1. Law of Continuing Change

This law states that any software system that represents some real-world reality undergoes continuous change or become progressively less useful in that environment.

2. Law of Increasing Complexity

As an evolving program changes, its structure becomes more complex unless effective efforts are made to avoid this phenomenon.

3. Law of Conservation of Organization Stability

Over the lifetime of a program, the rate of development of that program is approximately constant and independent of the resource devoted to system development.

4. Law of Conservation of Familiarity

This law states that during the active lifetime of the program, changes made in the successive release are almost constant.

Importance of Software Evolution

Software evolution is essential for several reasons, ensuring that software systems remain effective, relevant, and valuable over time. Here are the key reasons why software evolution is necessary:

- **Adapting to New Requirements:** As user needs and business environments change, software must be updated to meet new demands and remain useful.

- **Maintaining Security:** As evolution involves, regular updates are necessary to protect against new security threats and vulnerabilities.
- **Improving Performance:** Software can be optimized over time to run more efficiently and make better use of resources.
- **Ensuring Compatibility:** Software must evolve to stay compatible with new technologies, standards, and regulations.
- **Enhancing User Experience:** In this evolution, continuous improvement based on user feedback helps in providing a better and more intuitive user experience.

What are the factors where changes are needed in the software?
There are several factors that may require changes to be made in software:

Changes in business requirements or goals:
As a business evolves, its needs and goals may change, which may require changes to the software to support new features or capabilities.

Changes in technology:
As technology advances, software may need to be updated to take advantage of new capabilities or to maintain compatibility with new hardware or operating systems.

Bugs and defects:
Software may need to be modified to fix defects or errors that are discovered after it has been released.

Security vulnerabilities:
Changes may be needed to address security vulnerabilities that are discovered in the software.

Performance issues:
If the software is not performing as expected, changes may be needed to improve its performance.

User feedback:
Feedback from users may identify areas where the software can be improved or enhanced, which may require changes to be made.

Regulatory or compliance issues:
Changes may be required to ensure that the software meets regulatory or compliance requirements.

Types of Software Maintenance in Software Evolution

Software maintenance is the process of identifying and correcting defects in a software system, as well as making improvements and updates to the system to keep it running smoothly and efficiently. There are several types of software maintenance, including:

Corrective maintenance:

This type of maintenance is focused on fixing errors and defects in the software system. This includes fixing bugs, errors, and other issues that cause the system to malfunction or not work as intended.

Adaptive maintenance:

This type of maintenance involves making changes to the software system to make it compatible with new hardware, software, or operating systems. This may include updates to the system to support new features or functionality.

Perfective maintenance:

This type of maintenance is focused on improving the performance, functionality, and usability of the software system. This may include adding new features or functionality, optimising code, or improving the user interface.

Preventive maintenance:

This type of maintenance is focused on proactively addressing potential issues with the software system before they occur. This may include regular testing and debugging, as well as implementing measures to prevent future issues from occurring.

Components of Software Characteristics

There are 6 components of Software Characteristics are discussed here. We will discuss each one of them in detail.

Functionality:

It refers to the degree of performance of the software against its intended purpose.

Functionality refers to the set of features and capabilities that a software program or system provides to its users. It is one of the most important characteristics of software, as it determines the usefulness of the software for the intended purpose. Examples of functionality in software include:

- Data storage and retrieval
- Data processing and manipulation
- User interface and navigation
- Communication and networking

- Security and access control
- Reporting and visualization
- Automation and scripting

The more functionality a software has, the more powerful and versatile it is, but also the more complex it can be. It is important to balance the need for functionality with the need for ease of use, maintainability, and scalability.

Reliability:

A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.

Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time. Reliability is an important aspect of software quality, as it helps ensure that the software will work correctly and not fail unexpectedly.

Examples of factors that can affect the reliability of software include:

1. Bugs and errors in the code
2. Lack of testing and validation
3. Poorly designed algorithms and data structures
4. Inadequate error handling and recovery
5. Incompatibilities with other software or hardware

To improve the reliability of software, various techniques, and methodologies can be used, such as testing and validation, formal verification, and fault tolerance.

Software is considered reliable when the probability of it failing is low and it is able to recover from the failure quickly, if any.

Efficiency:

It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements.

Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way. High efficiency means that a software program can perform its intended functions quickly and with minimal use of resources, while low efficiency means that a software program may be slow or consume excessive resources.

Examples of factors that can affect the efficiency of the software include:

1. Poorly designed algorithms and data structures
2. Inefficient use of memory and processing power
3. High network latency or bandwidth usage
4. Unnecessary processing or computation
5. Unoptimized code

To improve the efficiency of software, various techniques, and methodologies can be used, such as performance analysis, optimization, and profiling.

Efficiency is important in software systems that are resource-constrained, high-performance, and real-time systems. It is also important in systems that need to handle many users or transactions simultaneously.

Usability:
It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

Maintainability:
It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

Portability:
A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Characteristics of "Software" in Software Engineering

1. **Software is developed or engineered; it is not manufactured in the classical sense:**

- Although some similarities exist between software development and hardware manufacturing, few activities are fundamentally different.
- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.

2. **The software doesn't "wear out.":**

- Hardware components suffer from the growing effects of many other environmental factors. Stated simply, the hardware begins to wear out.
- Software is not susceptible to the environmental maladies that cause hardware to wear out.
- When a hardware component wears out, it is replaced by a spare part.
- There are no software spare parts.
- Every software failure indicates an error in design or in the process through which the design was translated into machine-executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance. However, the implication is clear—the software doesn't wear out. But it does deteriorate.

3. **The software continues to be custom-built:**

- A software part should be planned and carried out with the goal that it tends to be reused in various projects.
- Current reusable segments encapsulate the two pieces of information and the preparation that is applied to the information, empowering the programmer to make new applications from reusable parts.
- In the hardware world, component reuse is a natural part of the engineering process.

Characteristics of the Software

- It is intangible, meaning it cannot be seen or touched.
- It is non-perishable, meaning it does not degrade over time.
- It is easy to replicate, meaning it can be copied and distributed easily.
- It can be complex, meaning it can have many interrelated parts and features.
- It can be difficult to understand and modify, especially for large and complex systems.
- It can be affected by changing requirements, meaning it may need to be updated or modified as the needs of users change.
- It can be impacted by bugs and other issues, meaning it may need to be tested and debugged to ensure it works as intended.

Application Software

The term "application software" refers to software that performs specific functions for a user. When a user interacts directly with a piece of software, it is called application software. The sole purpose of application software is to assist the user in doing specified tasks. Microsoft Word and Excel, as well as popular web browsers like Firefox and Google Chrome, are examples of application software. It also encompasses the category of mobile apps, which includes apps like WhatsApp for communication and games like Candy Crush Saga. There are also app versions of popular services, such as weather or transportation information, as well as apps that allow users to connect with businesses. Global Positioning System (GPS), Graphics, multimedia, presentation software, desktop publishing software, and so on are examples of such software.

Functions of Application Software

Application software programs are created to help with a wide range of tasks. Here are a few examples:

- Information and data management
- Management of documents (document exchange systems)
- Development of visuals and video
- Emails, text messaging, audio, and video conferencing, and cooperation are all options.
- Management of accounting, finance, and payroll
- Management of resources (ERP and CRM systems)
- Management of a project
- Management of business processes
- Software for education (LMS and e-learning systems)
- Software for healthcare applications

Application Software

Need for Application Software

End-users can use "application software" to conduct single or many tasks. Following are a few reasons to need application software in computers:

- **Helps the user in completing specified tasks:** Application software is designed with the user in mind. They help the end-user with specialized tasks in a variety of industries, including education, business, and entertainment. Microsoft Word, for example, is popular application software that allows users to create, edit, delete, and do other tasks with Word documents.
- **Manages and manipulates data:** Business companies utilize application software to manage and manipulate employees, customers, and other databases. Enterprise resource management systems and customer relationship management systems are two common examples of application software.
- **Allows users to effectively organize information:** Individual users can use application software to efficiently create and handle large amounts of data. Microsoft Excel, for example, is popular application software that allows users to manage datasheets.

Types of Application Software

Application software can also be categorized based on its charge ability and accessibility. Here is some application software:

- **Freeware:** It is offered for free, as the name implies. You can utilize freeware application software that you can obtain from the Internet. This software, on the other hand, does not allow you to change it or charge a fee for sharing it. Examples include Adobe PDF, Mozilla Firefox, and Google Chrome.
- **Shareware:** This is given away to users for free as a trial, usually with a limited-time offer. If consumers want to keep using this application software, they will have to pay. WinZip, Anti-virus, and Adobe Reader are instances of shareware.
- **Open-source:** This type of application software comes with the source code, allowing you to edit and even add features to it. These could be offered for free or for a fee. Open-source application software includes Moodle and Apache Web Server.
- **Closed source:** This category includes the majority of the application software programs used nowadays. These are normally charged, and the source code is usually protected by intellectual property rights or patents. It usually comes with a set of restrictions. Microsoft Windows, Adobe Flash Player, WinRAR, macOS, and other operating systems are examples.
- **Word Processing Software:** Word Processing Software can be explained as software that has the functionalities of editing, saving, and creating documents with Word Processor Software like Microsoft Word.
- **Spreadsheet Software:** Spreadsheet Software is a kind of software that deals with the worksheet where it works on some automated version to perform numeric functions. For Example, Microsoft Excel.
- **Presentation Software:** It is a type of application software that is used to present some applications like newly launched functions, products, etc. For Example, Microsoft Powerpoint.

- **Multimedia Software:** Multimedia refers to the mixture of audio, video, image, text, etc., and can be displayed or used with the help of multimedia software. There are so many media players that do this kind of work.
- **Web Browsers:** Web Browser is one of the most used applications worldwide, it takes you to the internet. You can use your desktop, mobile, etc for using this.
- **Educational Software:** Due to the enhancement of the Internet, there are so many educational software runs in the market. It consists of Language learning Software, Classroom Management Software, etc.
- **Graphics Software:** Graphics Software is also used in large amounts. There are so many applications where it is used. Some of the applications include Canva, Adobe, PhotoShop, etc.
- **Simulation Software:** Simulation Software is a kind of Software that is used to compare two different kinds of products and also it helps in evaluating them.

A Crisis on the Horizon?

The **Software Crisis** refers to a period—starting in the late 1960s—when the **demand for complex, reliable, and large-scale software systems** rapidly increased. Unfortunately, the software industry was **unprepared** for this growth. As a result, many projects failed to meet user expectations, and issues like **poor quality**, **missed deadlines**, and **escalating costs** became alarmingly common.

Key Issues That Fueled the Crisis

Let's look at the main contributors to this crisis:

1. *Missed Deadlines*

Many software projects **took longer** than originally planned. Due to poor estimation and lack of formal planning techniques, delivery timelines were **constantly extended**, leading to **project delays** and dissatisfied stakeholders.

2. *Cost Overruns*

Budgets were often **underestimated**, and as projects progressed, the cost of resources, changes in requirements, and bug fixes caused **expenditures to spiral** out of control.

3. *Poor Quality*

A common symptom of the software crisis was software that was **buggy, unreliable, and hard to use**. Frequent system crashes, security flaws, and missing features made such products **unacceptable** in professional environments.

4. *Unmaintainable Code*

Early software lacked **modular design**, documentation, and coding standards. As a result, when bugs were found or changes were needed, **developers couldn't easily understand or modify** existing code—leading to "spaghetti code."

Software Myths- Software Engineering

In software engineering, myths about software and its creation persist despite being untrue. These false beliefs have become common knowledge among managers and developers, making breaking free from old habits challenging.

To unleash the true potential of software development, it's crucial to identify and dispel these long-standing misconceptions. By doing so, we can ensure that decisions are grounded in reality and best practices, enabling us to achieve greater efficiency and success in our projects. Embracing a fresh perspective and challenging these myths will pave the way for a new era of innovation and progress in the world of software engineering.

Myths About Software Development

1. Software Development Comes with a Hefty Price Tag

Perhaps this is the **most popular myth about software development.** It is because of this myth companies do not harness the potential that custom software can provide which can improve their organization's efficiency. Instead, they opt for purchasing some "one size fits all" solution which, of course, doesn't fit their requirements and they have to find other means to work around it. Also, consider the investment loss if the company outgrows the software and it just becomes unworkable. If you combine this with hidden costs such as upgrade fees, licensing, and support costs, custom software doesn't seem so expensive.

2. Users Have No Idea What They Know and Want

Even though there conversely exists a myth that customers have no idea what they want until you show them, regardless, software companies need to be both product-oriented and customer oriented as long as they consider the speed of delivery to be important. Wise businesses don't simply focus on creating a top-notch, most innovative product on the market, but they also strive to deliver the best solution to the consumer, that provides market success. Doing no market research and not listening to your customers is simply not an option.

3. The Waterfall Method Still Works

You would be amazed at the number of people who still believe that a system can be specified in detail before you even build it. Not only is this almost impossible, but it is also inefficient to execute the development process in a sequence. While there also exists a myth that Agile lacks any planning whatsoever, the fact of the matter is that planning is just as necessary to the effectiveness of Agile as it is to Waterfall, but the difference is in the way the planning is done. Waterfall promotes planning before building at the very beginning of the project which poses a lot of constraints in flexibility and adaptation. Conversely, Agile allows for an ongoing planning mechanism where changes and adjustments are made as the project goes along in an iterative manner.

4. The More, The Merrier

Unfortunately, a myth exists that adding people to a development team makes it better and speeds up delivery. However, adding more people to a project tends to prolong the project's timeframe and causes friction due to issues in training and collaboration.

5. Software Development Has a Fixed Cost and Strict Timeframe

Going back to the Waterfall approach, it is simply not possible to detail the software before building it. Even though a lot of companies are lured in by the fixed price model, they must also remember that there are some hidden costs associated with it in terms of quality and additional costs incurred as the project goes beyond the projected time boundaries.

6. There's Always a "Magic Bullet"

Just like the old adage "A bad worker blames his tools", many people believe that they are missing some state-of-the-art tool that will solve all of their problems and produce magnificent results. When building high-quality software, the utmost importance should be given to critical innovative thinking, agility, and skillset. Having the best technologies is just the icing on the cake.

7. When the Software is Released, the Project is Over

As soon as the product is released, the focus should be on receiving feedback from the users and incorporating this feedback into an iterative approach back into the product. There needs to be an ongoing process of improvement and revisions along with testing for bugs in order to provide the customer with the best quality product.

8. Product Owners and Developers Work in a Single Location

Thanks to modern communication technology, distributed teams can use collaborative tools such as e-mail, shared calendars, instant messengers, screen sharing, audio and video conferencing, and many others to work together and stay up to date. Nowadays companies have access to a wide array of communication tools that are widely accessible and readily available thus allowing remote developers to work seamlessly.

9. Outsourcing Solves Everything

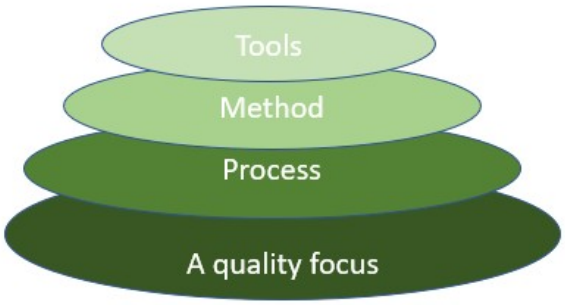
Plain old outsourcing can create a far worse problem than you initially had. Software vendors should start thinking about establishing long-term relationships and engaging in a dedicated team model instead of fixed-cost projects which lead to incomplete deliverables, frustration, and high overhead costs.

10. Outsourcing Means Compromising Quality

When done properly, and in a well-researched manner, outsourcing can provide you with better code and an outstanding product. When you go offshore, you gain access to a wide talent pool that has the industry expertise that you have been struggling to fill in-house. Do not hesitate to explore new strategies!

Layered Technology in Software Engineering

Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

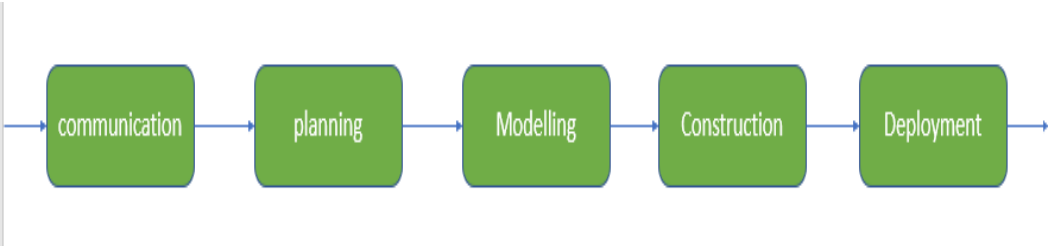


Just as software engineering requires progressing through interconnected layers to build robust software, advancing your skills in software testing also involves a step-by-step approach. To effectively move from basic testing to more complex automation, consider exploring.This course will help you build on each layer of your knowledge, ensuring you master the intricacies of testing and automation to create reliable, high-quality software.

Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.



Process activities are listed below:-

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modeling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment:-** It includes the delivery of software to the client for evaluation and feedback.

3. Method: During the process of software development the answers to all "how-to-do" questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Software Processes in Software Engineering

Software Processes

Software processes in software engineering refer to the methods and techniques used to develop and maintain software. Some examples of software processes include:

- **Waterfall**: a linear, sequential approach to software development, with distinct phases such as requirements gathering, design, implementation, testing, and maintenance.
- **Agile**: a flexible, iterative approach to software development, with an emphasis on rapid prototyping and continuous delivery.
- **Scrum**: a popular Agile methodology that emphasizes teamwork, iterative development, and a flexible, adaptive approach to planning and management.
- **DevOps**: a set of practices that aims to improve collaboration and communication between development and operations teams, with an emphasis on automating the software delivery process.

Each process has its own set of advantages and disadvantages, and the choice of which one to use depends on the specific project and organization.

Components of Software

There are three main components of the software:

1. **Program:** A computer program is a list of instructions that tell a computer what to do.
2. **Documentation:** Source information about the product contained in design documents, detailed code comments, etc.
3. **Operating Procedures:** Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.

Other Software Components

Other Software Components are:

1. **Code**: the instructions that a computer executes in order to perform a specific task or set of tasks.
2. **Data**: the information that the software uses or manipulates.
3. **User** interface: the means by which the user interacts with the software, such as buttons, menus, and text fields.
4. **Libraries**: pre-written code that can be reused by the software to perform common tasks.
5. **Documentation**: information that explains how to use and maintain the software, such as user manuals and technical guides.
6. **Test cases**: a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.

7. **Configuration files**: files that contain settings and parameters that are used to configure the software to run in a specific environment.
8. **Build and deployment** scripts: scripts or tools that are used to build, package, and deploy the software to different environments.
9. **Metadata**: information about the software, such as version numbers, authors, and copyright information.

Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

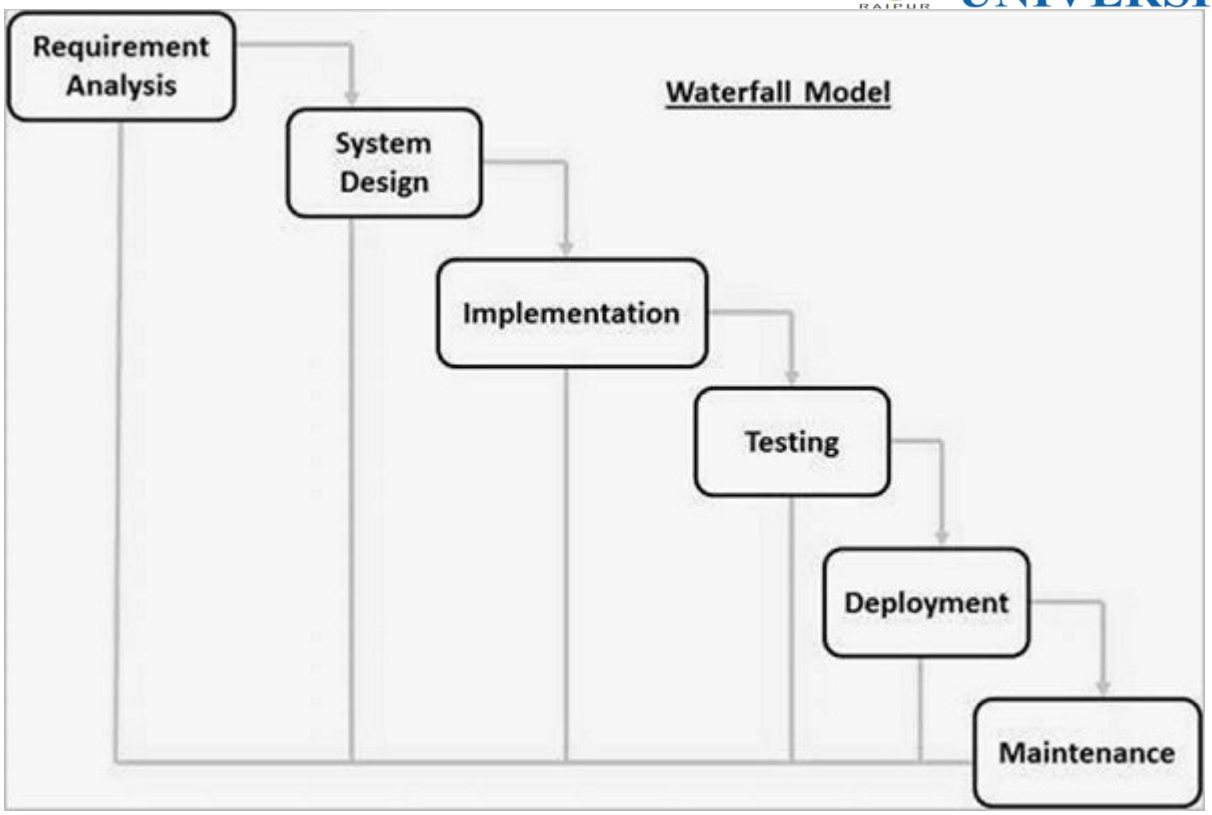
The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

Requirements are very well documented, clear and fixed.

Product definition is stable.

Technology is understood and is not dynamic.

There are no ambiguous requirements.

Ample resources with required expertise are available to support the product.

The project is short.

Waterfall Model - Advantages

- Some of the major advantages of the Waterfall Model are as follows −
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

- The major disadvantages of the Waterfall Model are as follows −
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.

- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

RAD Model

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

What is RAD?

Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Following are the various phases of the RAD Model −

Business Modelling

The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modelling

The information gathered in the Business Modelling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modelling

The data object sets defined in the Data Modelling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

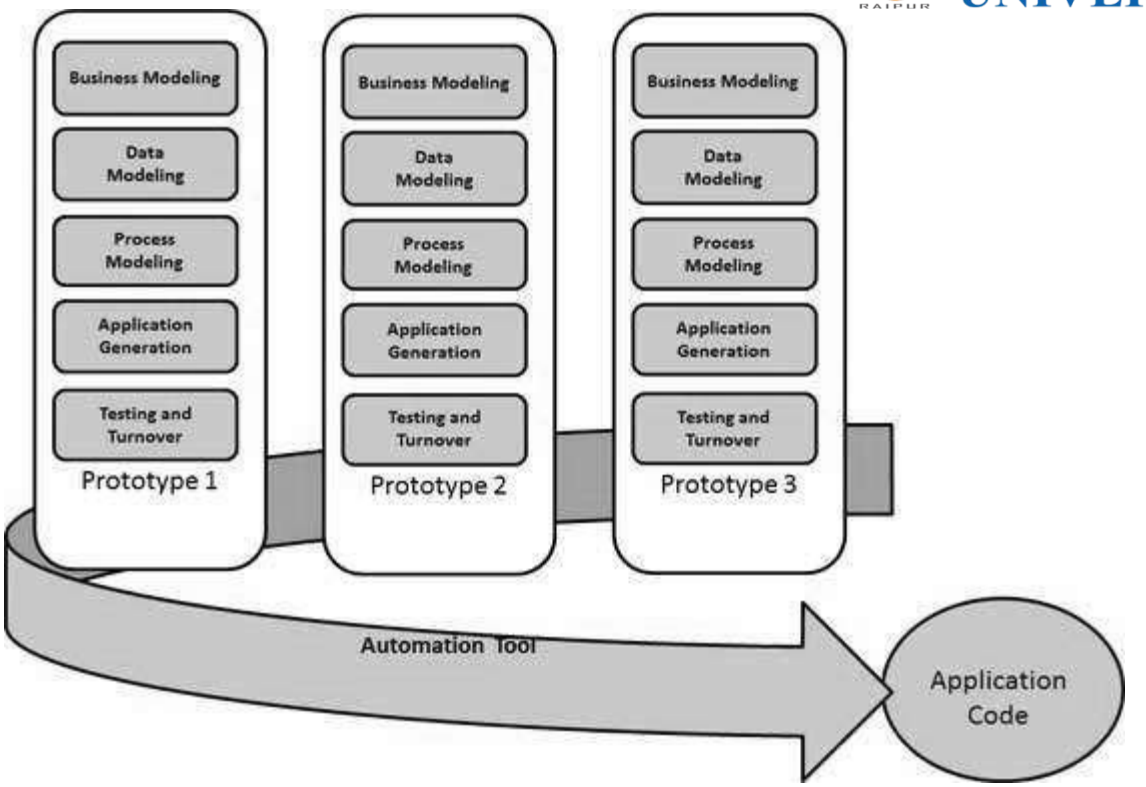
Application Generation

The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

Testing and Turnover

The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

The following illustration describes the RAD Model in detail.



RAD Model - Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail.

The following pointers describe the typical scenarios where RAD can be used −

RAD should be used only when a system can be modularized to be delivered in an incremental manner.

It should be used if there is a high availability of designers for Modelling.

It should be used only if the budget permits use of automated code generating tools.

RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.

Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model - Advantages

The advantages of the RAD Model are as follows −

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.

- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

RAD Model - Disadvantages

The disadvantages of the RAD Model are as follows −

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on Modelling skills.
- Inapplicable to cheaper projects as cost of Modelling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.

Software Prototype Model

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is a stepwise approach explained to design a software prototype.

Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and Enhance the Prototype

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Prototypes can have horizontal or vertical dimensions. A Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A Vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping - Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely −

Throwaway/Rapid Prototyping

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. By using evolutionary prototyping, the well-understood requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping

Incremental prototyping refers to building multiple functional prototypes of the various sub-systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the HTML format. Then the data processing is simulated using a prototype services layer. Finally, the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping - Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping – Advantages

The advantages of the Prototyping Model are as follows −

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

Software Prototyping – Advantages

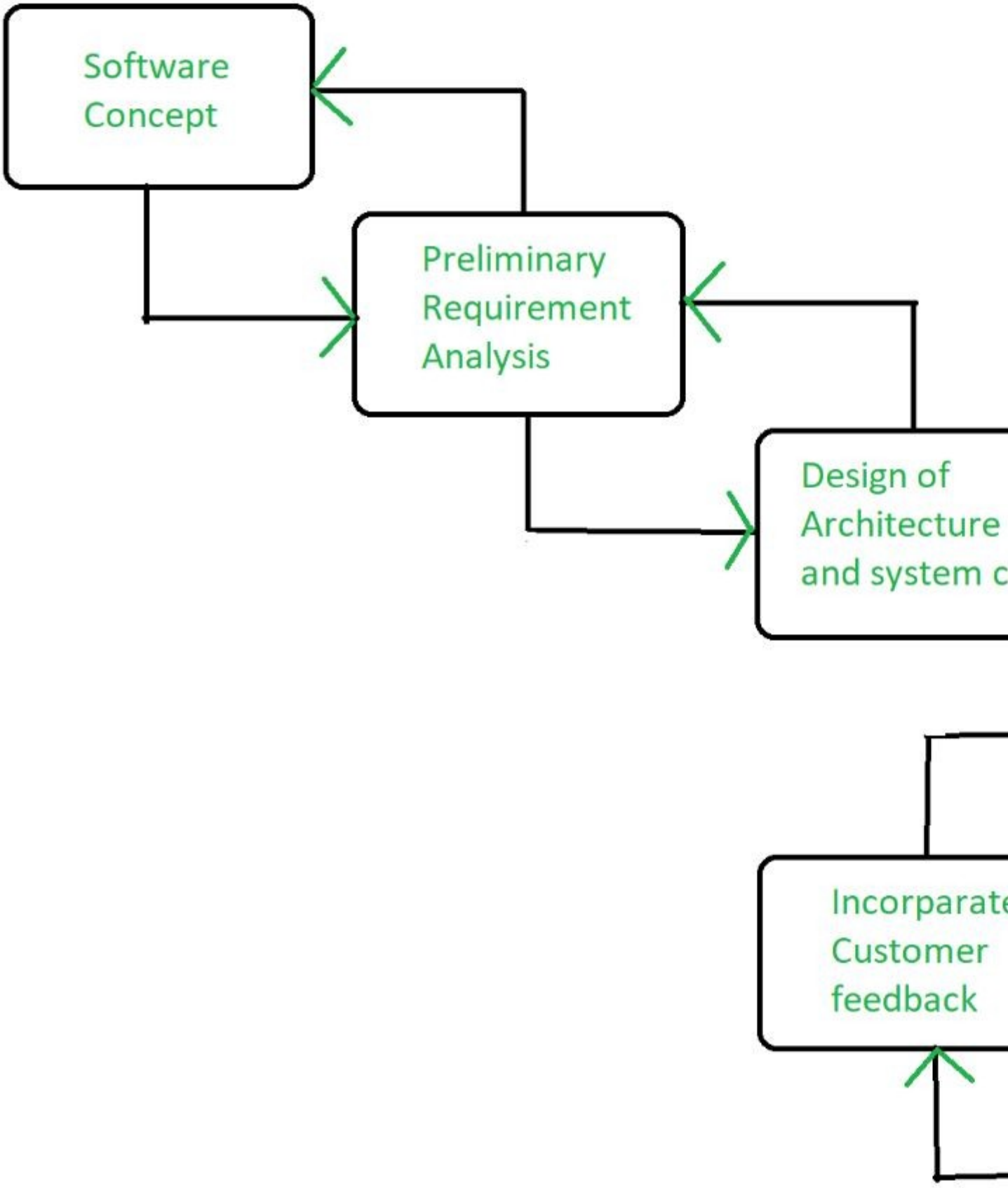
The Disadvantages of the Prototyping Model are as follows −

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

Evolutionary Model

The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users can get access to the product at the end of each cycle.

1. Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan, or process.
2. Therefore, the software product evolves with time.
3. All the models have the disadvantage that the duration of time from the start of the project to the delivery time of a solution is very high.
4. The evolutionary model solves this problem with a different approach.
5. The evolutionary model suggests breaking down work into smaller chunks, prioritizing them, and then delivering those chunks to the customer one by one.
6. The number of chunks is huge and is the number of deliveries made to the customer.
7. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements.
8. The model allows for changing requirements as well as all work is broken down into maintainable work chunks.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system c
Incorparate Customer feedback

Evolutionary Model

Application of Evolutionary Model

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Necessary Conditions for Implementing this Model

1. Customer needs are clear and been explained in deep to the developer team.
2. There might be small changes required in separate parts but not a major change.
3. As it requires time, so there must be some time left for the market constraints.
4. Risk is high and continuous targets to achieve and report to customer repeatedly.
5. It is used when working on a technology is new and requires time to learn.

Advantages Evolutionary Model

1. **Adaptability to Changing Requirements:** Evolutionary models work effectively in projects when the requirements are ambiguous or change often. They support adjustments and flexibility along the course of development.
2. **Early and Gradual Distribution:** Functional components or prototypes can be delivered early thanks to incremental development. Faster user satisfaction and feedback may result from this.
3. **User Commentary and Involvement:** Evolutionary models place a strong emphasis on ongoing user input and participation. This guarantees that the software offered closely matches the needs and expectations of the user.
4. **Improved Handling of Difficult Projects:** Big, complex tasks can be effectively managed with the help of evolutionary models. The development process is made simpler by segmenting the project into smaller, easier-to-manage portions.

Disadvantages Evolutionary Model

1. **Communication Difficulties:** Evolutionary models require constant cooperation and communication. The strategy may be less effective if there are gaps in communication or if team members are spread out geographically.
2. **Dependence on an Expert Group:** A knowledgeable and experienced group that can quickly adjust to changes is needed for evolutionary models. Teams lacking experience may find it difficult to handle these model's dynamic nature.
3. **Increasing Management Complexity:** Complexity can be introduced by organizing and managing several increments or iterations, particularly in large projects. In order to guarantee integration and synchronization, good project management is needed.
4. **Greater Initial Expenditure:** As evolutionary models necessitate continual testing, user feedback and prototyping, they may come with a greater starting cost. This may be a problem for projects that have limited funding.

Component Based Development

Component-based development (CBD) is defined as a set of reuse-enabling technologies, tools and techniques that allow application development (AD) organizations to go through the entire AD process (i.e., analysis design, construction and assembly) or through any particular stage via the use of predefined component-enabling technologies (such as AD patterns, frameworks, design templates) tools and application building blocks.