

SOFTWARE ENGINEERING

Unit- I Introduction: Evolution, Software development projects, Exploratory style of software developments, Emergence of software engineering, Notable changes in software development practices, Computer system engineering.

Software Life Cycle Models: Basic concepts, Waterfall model and its extensions, Rapid application development, Agile development model, Spiral model.

IEEE Definition

- “**Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”**

Software Engineering: software engineering the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software

Characteristics of software:

- ✓ **Efficiency:** It provides a measure of the resource requirement of a software product efficiently.
- ✓ **Reliability:** It assures that the product will deliver the same results when used in similar working environment.
- ✓ **Reusability:** This attribute makes sure that the module can be used in multiple applications.
- ✓ **Maintainability:** It is the ability of the software to be modified, repaired, or enhanced easily with changing requirements.
- ✓ **Portability:**

Evolution

Software Evolution is a term that refers to the process of developing software initially, and then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities

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

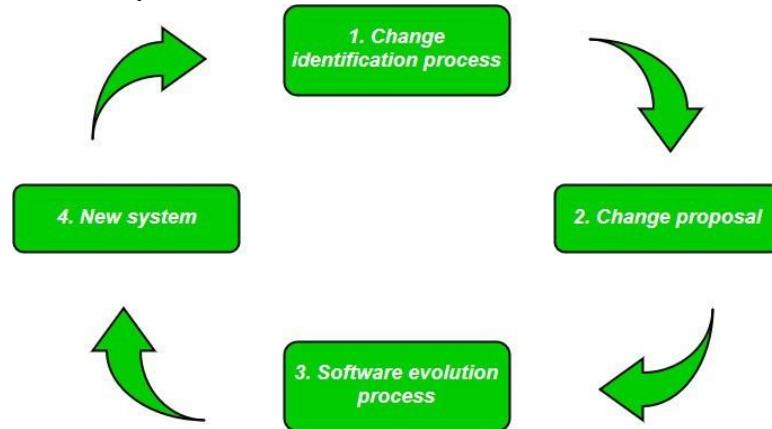
1. The cost and impact of these changes are accessed to see how much the system is affected by the change and how much it might cost to implement the change.
2. If the proposed changes are accepted, a new release of the software system is planned.

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.



Software development projects

Here are nine steps you can follow to develop a software project:

1. Evaluate your project. Assess the feasibility of your project. This includes analyzing its concept, goals, scope and specifications.
2. Specify the requirements. Figure out and document the technical needs of the project. During this phase, it's important to answer questions about who the target user is and what this project solves or makes easier for them.
3. Make a plan. Determine the specific components of the project and make a timeline for the completion of each task. This includes figuring out which tasks depend on others for completion, what resources you need and what your budgetary requirements are.
4. Conceptualize the design. Design the project's architecture and functions. Software architects and engineers typically handle this stage of the software development life cycle.
5. Establish metrics. Set up software metrics to help you continuously track and evaluate your project's progress. Consider using project analytics software to help you regularly collect and analyze your data.
6. Develop the software. Develop and code the project. This step is usually one of the longer phases of the software development life cycle.
7. Conduct testing. Rigorously test and conduct quality control assessments of the software. During the testing phase, it's important to evaluate various aspects of the software, including the quality of the code, compliance with regulations and how well the software meets the established requirements or metrics.
8. Deploy the software. Deploy the software to production. Depending on factors like how complex the software is, you might deploy the program several times, such as deploying it first for pre-release testing.
9. Perform post-production tasks. Conduct updates or maintenance work as needed on the software. This may include updating the software to work on new operating systems, increasing its scale or adding new features.

Changing Nature of Software

The software is an instruction or computer program that when executed provides desired features, function, and performance. A data structure that enables the program to adequately manipulate information and documents that describe the operation and use of the program.

Nowadays, seven broad categories of computer software present continuing challenges for software engineers.

1. **System Software:** System software is a collection of programs that are written to service other programs. Some system software processes complex but

determinate, information structures. Other system application processes largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.

2. **Application Software:** Application software is defined as programs that solve a specific business need. Application in this area processes business or technical data in a way that facilitates business operation or management technical decision-making. In addition to conventional data processing applications, application software is used to control business functions in real-time.
3. **Engineering and Scientific Software:** This software is used to facilitate the engineering function and task. however modern applications within the engineering and scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.
4. **Embedded Software:** Embedded software resides within the system or product and is used to implement and control features and functions for the end-user and for the system itself. Embedded software can perform limited and esoteric functions or provide significant function and control capability.
5. **Product-line Software:** Designed to provide a specific capability for use by many customers, product-line software can focus on the limited and esoteric marketplace or address the mass consumer market.
6. **Web Application:** It is a client-server computer program that the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B applications grow in importance. Web apps are evolving into a sophisticated computing environment that not only provides a standalone feature, computing function, and content to the end user.
7. **Artificial Intelligence Software:** Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

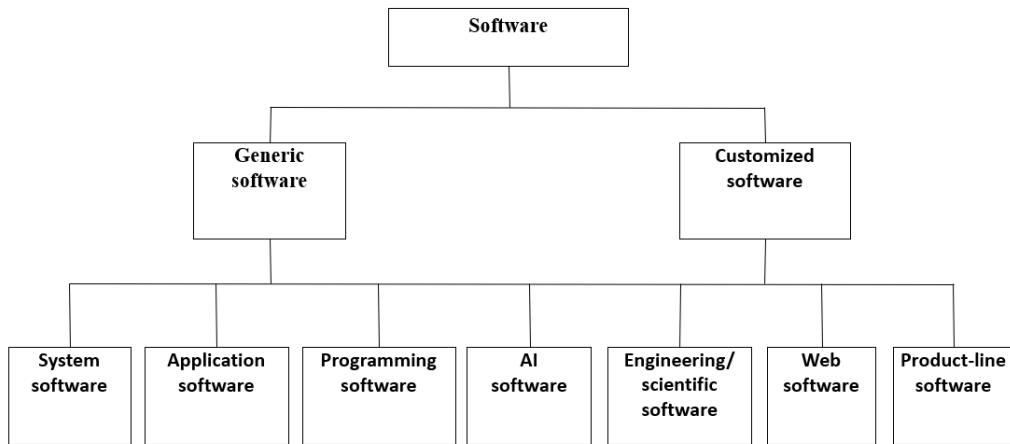


Figure 1.2: Software classification

EXPLORATORY STYLE OF SOFTWARE DEVELOPMENT

The exploratory program development style refers to an informal development style where the programmer makes use of his own intuition to develop a program rather than making use of the systematic body of knowledge categorized under the software engineering discipline.

The exploratory development style gives complete freedom to the programmer to choose the activities using which to develop software

the developers start coding to develop a working program. The software is tested and the bugs found are fixed. This cycle of testing and bug fixing continues till the software works satisfactorily for the customer

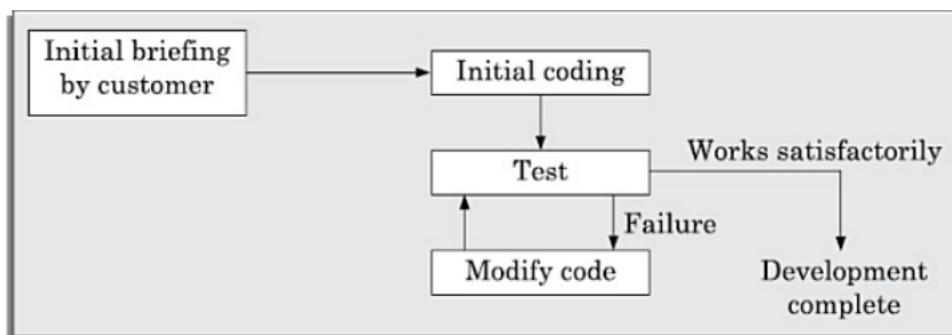


Figure 1.3: Exploratory program development.

An exploratory development style can be successful when used for developing very small programs, and not for professional software.

What is wrong with the exploratory style of software development?

In an exploratory development scenario, let us examine how do the effort and time required to develop a professional software increases with the increase in program size.

Let us first consider that exploratory style is being used to develop a professional software. The increase in development effort and time with problem size has been indicated in Figure 1.4

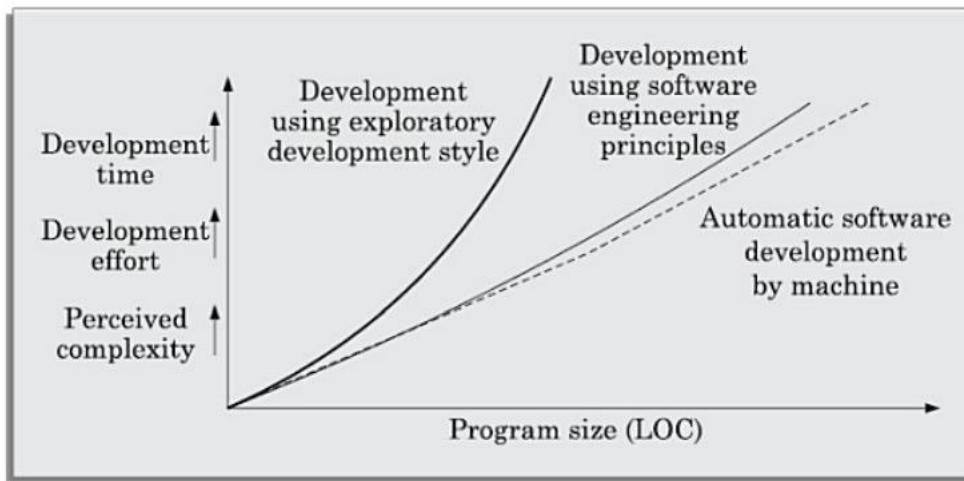


Figure 1.4: Increase in development time and effort with problem size.

the thick line plot that represents the case in which the exploratory style is used to develop a program. It can be seen that as the program size increases, the required effort and time increases almost exponentially.

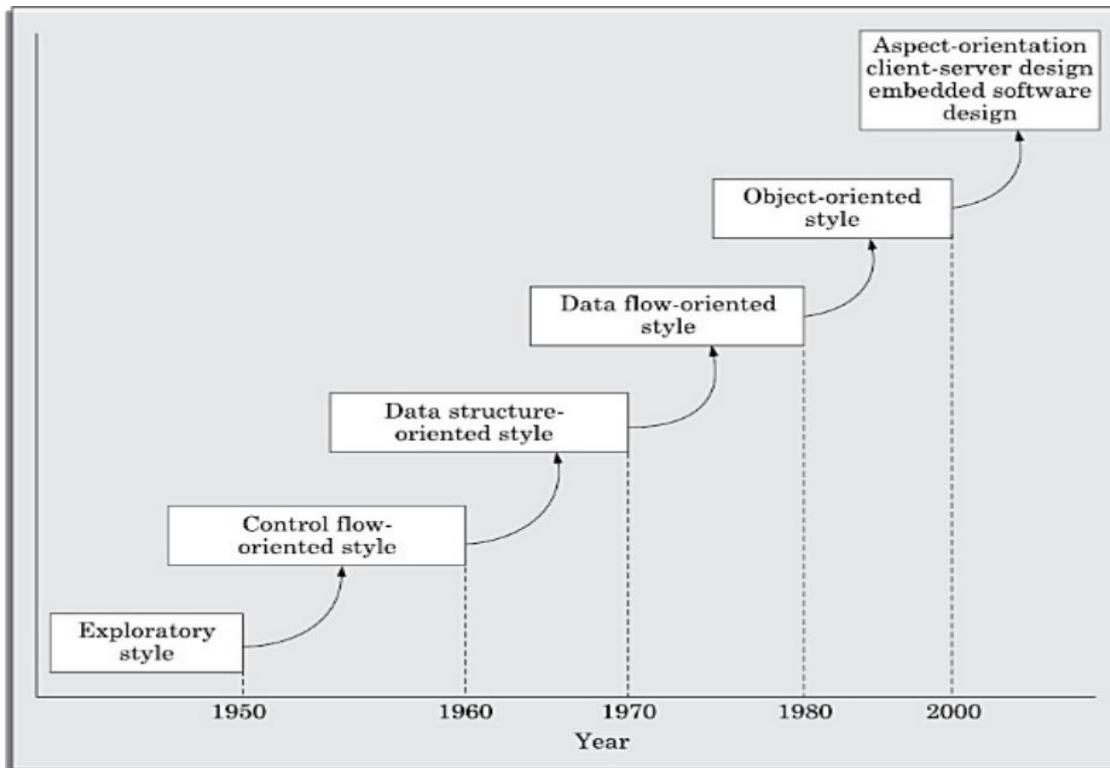
the thin solid line plot in Figure 1.4 which represents the case when development is carried out using software engineering principles. In this case, it becomes possible to solve a problem with effort and time that is almost linear in program size. On the other hand, if programs could be written automatically by machines, then the increase in effort and time with size would be even closer to a linear (dotted line plot) increase with size.

shortcomings of the exploratory style of software development:

- The foremost difficulty is the exponential growth of development time and effort with problem size and large-sized software becomes almost impossible using this style of development.
- The exploratory style usually results in unmaintainable code. The reason for this is that any code developed without proper design would result in highly unstructured and poor-quality code.
- It becomes very difficult to use the exploratory style in a team development environment. In the exploratory style, the development work is undertaken without any proper design and documentation.

EMERGENCE OF SOFTWARE ENGINEERING

This evolution is the result of a series of innovations and accumulation of experience about writing good quality programs. Since these innovations and programming experiences are too numerous, let us briefly examine only a few of these innovations and programming experiences which have contributed to the development of the software engineering discipline.



style of programming as the build and fix (or the exploratory programming) style.

High-level Language Programming

high-level languages such as FORTAN, ALGOL, and COBOL were introduced in 1960's. This considerably reduced the effort required to develop software and helped programmers to write larger programs. However, programmers were still using the exploratory style of software development. Typical programs were limited to sizes of around a few thousands of lines of source code.

Control Flow-based Design

experienced programmers advised other programmers to pay particular attention to the design of a program's control flow structure

program's instructions are executed. In order to help develop programs having good control flow structures, the flow charting technique was developed. Even today, the flow charting technique is being used to represent and design algorithms

Structured programming—a logical extension

A program is called structured when it uses only the sequence, selection, and iteration types of constructs and is modular.

The need to restrict the use of GO TO statements was recognized by everybody and make use of implementation of data structures in the code.

Gradually, everyone accepted that it is indeed possible to solve any programming problem without using GO TO statements and that indiscriminate use of GO TO statements should be avoided. This formed the basis of the structured programming methodology.

Data Structure-oriented Design

Using data structure-oriented design techniques, first a program's data structures are designed. The code structure is designed based on the data structure.

Data Flow-oriented Design

The data flow-oriented techniques advocate that the major data items handled by a system must

be identified and the processing required on these data items to produce the desired outputs should be determined.

Object-oriented Design

Data flow-oriented techniques evolved into object-oriented design (OOD) techniques in the late seventies. Object-oriented design technique is an intuitively appealing approach, where the natural objects (such as employees, pay-roll-register, etc.) relevant to a problem are first identified and then the relationships among the objects such as composition, reference, and inheritance are determined.

NOTABLE CHANGES IN SOFTWARE DEVELOPMENT PRACTICES

the glaring differences that you would notice when you observe an exploratory style of software development and another development effort based on modern software engineering practices.

- An important difference is that the exploratory software development style is based on error correction (build and fix) while the software engineering techniques are based on the principles of error prevention.
- In the exploratory style, coding was considered synonymous with software development. Exploratory programmers literally dive at the computer to get started with their programs even before they fully learn about the problem!!!
In the modern software development style, coding is regarded as only a small part of the overall software development activities.
- A lot of attention is now being paid to requirements specification. Significant effort is being devoted to develop a clear and correct specification of the problem before any development activity starts. Unless the requirements specification is able to correctly capture the exact customer requirements, large number of rework would be necessary at a later stage
- Periodic reviews are being carried out during all stages of the development process. The main objective of carrying out reviews is phase containment of errors, i.e. detect and correct errors as soon as possible.
- Today, software testing has become very systematic and standard testing techniques are available.
- In the past, very little attention was being paid to producing good quality and consistent documents. In the exploratory style, the design and test activities, even if carried out (in whatever way), were not documented satisfactorily. Today, consciously good quality documents are being developed during software development.

COMPUTER SYSTEMS ENGINEERING

Computer system engineering encompasses the design, development, and management of computer hardware systems. It is a multidisciplinary field that combines elements of electrical engineering, computer science, and computer engineering. Computer system engineering requires a deep understanding of both hardware and software to create robust and efficient computer systems.

It also involves the integration of hardware components, such as processors, memory, and storage devices, to create efficient and functional computer systems. The role of a computer system engineer is crucial in ensuring that computer systems are not only

reliable and secure but also capable of meeting the ever-increasing demands of today's technology-driven world.

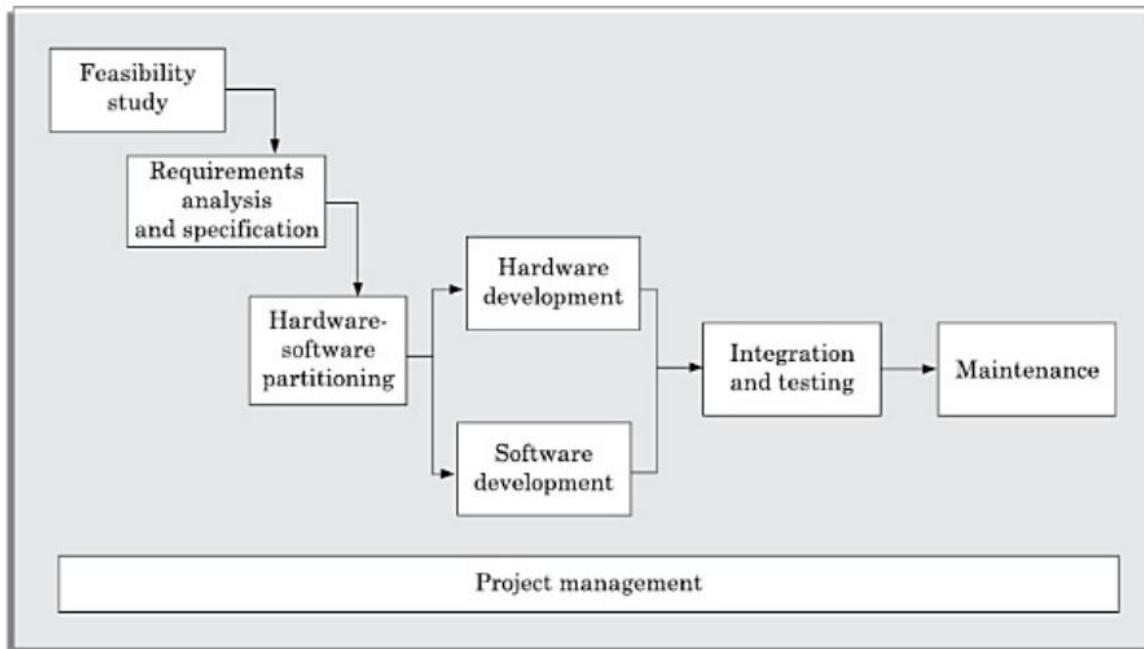


Figure 1.13: Computer systems engineering.

- The functionality implemented in hardware run faster. On the other had, functionalities implemented in software is easier to extend.
- After the hardware-software partitioning stage, development of hardware and software are carried out concurrently (shown as concurrent branches in Figure 1.13).
- In system engineering, testing the software during development becomes a tricky issue, the hardware on which the software would run and tested would still be under development—remember that the hardware and the software are being developed at the same time.
- To test the software during development, it usually becomes necessary to develop simulators that mimic the features of the hardware being developed.
- Once both hardware and software development are complete, these are integrated and tested.

UNIT 1 (PART-2) SOFTWARE LIFE CYCLE MODELS

BASIC CONCEPTS

Software life cycle

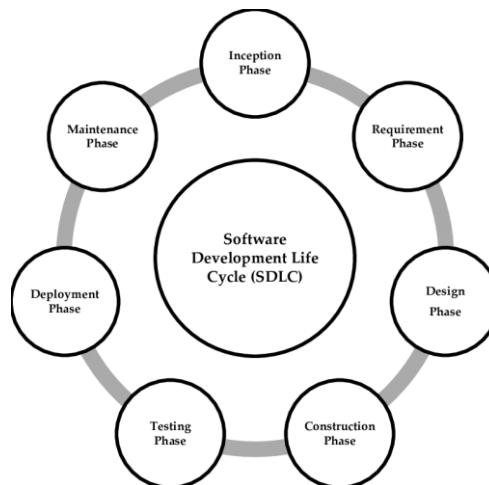
Based on this concept of a biological life cycle, the term software life cycle has been defined to imply the different stages (or phases) over which a software evolves from an initial customer request for it, to a fully developed software, and finally to a stage where it is no longer useful to any user, and then it is discarded.

The life cycle of a software represents the series of identifiable stages through which it evolves during its life time.

Software development life cycle (SDLC) model

A software development life cycle (SDLC) model (also called software life cycle model and software development process model) describes the different activities that need to be carried out for the software to evolve in its life cycle.

An SDLC graphically depicts the different phases through which a software evolves. It is usually accompanied by a textual description of the different activities that need to be carried out during each phase.



WATERFALL MODEL AND ITS EXTENSIONS

The waterfall model and its derivatives were extremely popular in the 1970s. The waterfall model is possibly the most obvious and intuitive way in which software can be developed through team effort.

Classical Waterfall Model

Classical waterfall model is intuitively the most obvious way to develop software. It is simple but idealistic.

In fact, it is hard to put this model into use in any non-trivial software development project. The classical waterfall model divides the life cycle into a set of phases as shown in Figure 2.1. It can be easily observed from this figure that the diagrammatic representation of the classical waterfall model resembles a multi-level waterfall. This resemblance justifies the name of the model.

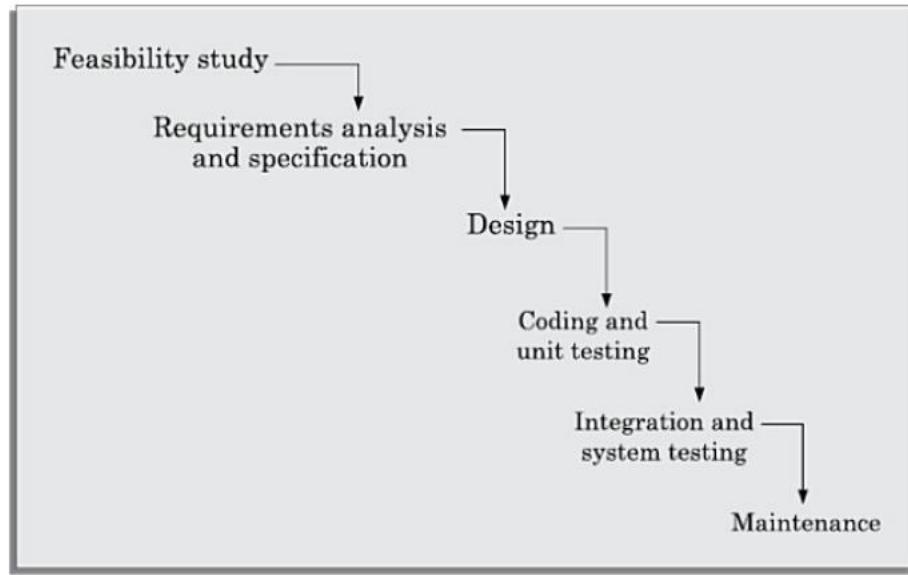


Figure 2.1: Classical waterfall model.

Phases of the classical waterfall model

A software is developed during the development phases, and at the completion of the development phases, the software is delivered to the customer.

An activity that spans all phases of software development is project management. Since it spans the entire project duration, no specific phase is named after it.

In the waterfall model, different life cycle phases typically require relatively different amounts of efforts to be put in by the development team. The relative amounts of effort spent on different phases for a typical software has been shown in Figure 2.2

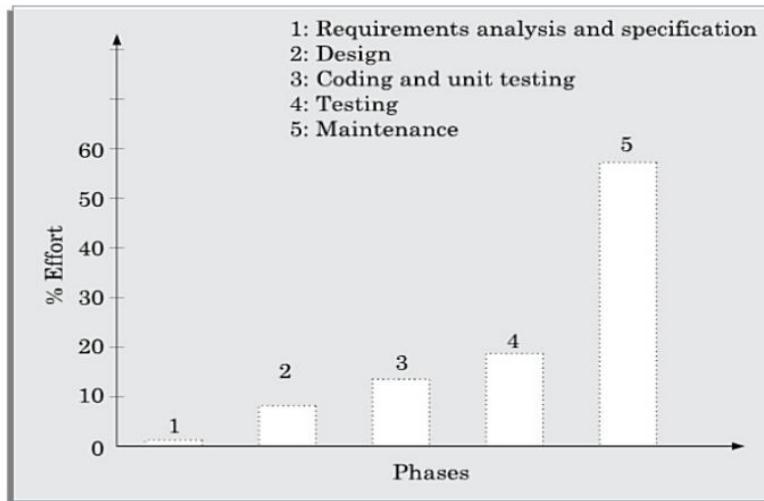


Figure 2.2: Relative effort distribution among different phases of a typical product.

we briefly describe the activities that are carried out in the different phases of the classical waterfall model.

1. Feasibility study

- The main focus of the feasibility study stage is to determine whether it would be financially and technically feasible to develop the software.
- feasibility study involves carrying out several activities such as collection of basic information relating to the software such as the different data items that would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system, as well as various

constraints on the development

These collected data are analyzed to perform at the following:

- i. **Development of an overall understanding of the problem:** It is necessary to first develop an overall understanding of what the customer requires to be developed.
- ii. **Formulation of the various possible strategies for solving the problem:** In this activity, various possible high-level solution schemes to the problem are determined.
- iii. **Evaluation of the different solution strategies:** The different identified solution schemes are analyzed to evaluate their benefits and shortcomings.

2. Requirements analysis and specification

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly.

Requirements gathering and analysis: The goal of the requirements' gathering activity is to collect all relevant information regarding the software to be developed from the customer with a view to clearly understand the requirements.

Requirements specification: After the requirement gathering and analysis activities are complete, the identified requirements are documented. This is called a software requirements specification (SRS) document. The SRS document is written using end-user terminology. This makes the SRS document understandable to the customer. Therefore, understandability of the SRS document is an important issue.

3. Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

Procedural design approach: The traditional design approach is in use in many software development projects at the present time. This traditional design technique is based on the data flow-oriented design approach.

Object-oriented design approach: In this technique, various objects that occur in the problem domain and the solution domain are first identified and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design. The OOD approach is credited to have several benefits such as lower development time and effort, and better maintainability of the software.

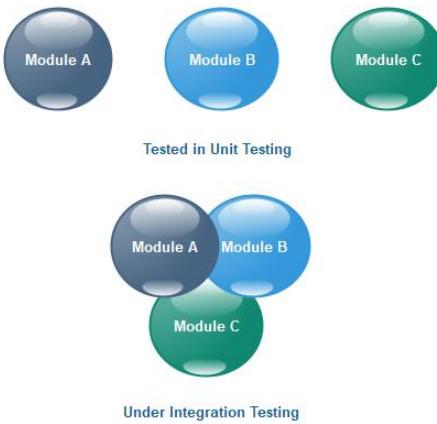
4. Coding and unit testing

- The purpose of the coding and unit testing phase is to translate a software design into source code and to ensure that individually each function is working correctly.
- The coding phase is also sometimes called the implementation phase, since the design is implemented into a workable solution in this phase

5. Integration and system testing

Integration of different modules is undertaken soon after they have been coded and unit tested. During the integration and system testing phase, the different modules are integrated in a planned manner

Integration testing is carried out to verify that the interfaces among different units are working satisfactorily. On the other hand, the goal of system testing is to ensure that the developed system conforms to the requirements that have been laid out in the SRS document.



6. Maintenance

The total effort spent on maintenance of a typical software during its operation phase is much more than that required for developing the software itself. Many studies carried out in the past confirm this and indicate that the ratio of relative effort of developing a typical software product and the total effort spent on its maintenance is roughly 40:60. Maintenance is required in the following three types of situations:

Corrective maintenance: This type of maintenance is carried out to correct errors that were not discovered during the product development phase.

Perfective maintenance: This type of maintenance is carried out to improve the performance of the system, or to enhance the functionalities of the system based on customer's requests.

Adaptive maintenance: Adaptive maintenance is usually required for porting the software to work in a new environment.

Shortcomings of the classical waterfall model

1. **No feedback paths:** In classical waterfall model, the evolution of a software from one phase to the next is analogous to a waterfall. Just as water in a waterfall after having flowed down cannot flow back, once a phase is complete, the activities carried out in it and any artifacts produced in this phase are considered to be final and are closed for any rework.
2. **Difficult to accommodate change requests:** This model assumes that all customer requirements can be completely and correctly defined at the beginning of the project.
3. **Inefficient error corrections:** This model defers integration of code and testing tasks until it is very late when the problems are harder to resolve.
4. **No overlapping of phases:** This model recommends that the phases be carried out sequentially—new phase can start only after the previous one completes

Iterative Waterfall Model

The main change brought about by the iterative waterfall model to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases.

The feedback paths introduced by the iterative waterfall model are shown in Figure 2.3. The feedback paths allow for correcting errors committed by a programmer during some phase, as and when these are detected in a later phase

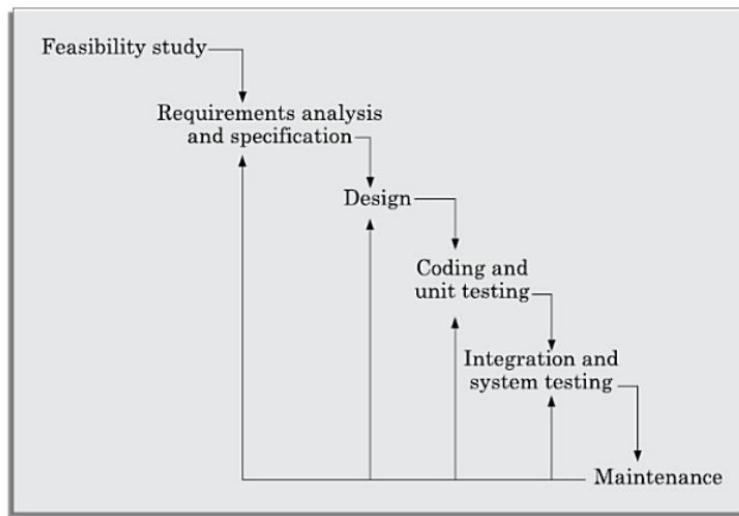


Figure 2.3: Iterative waterfall model.

Phase containment of errors No matter how careful a programmer may be, he might end up committing some mistake or other while carrying out a life cycle activity. These mistakes result in errors (also called faults or bugs) in the work product.

The principle of detecting errors as close to their points of commitment as possible is known as *phase containment of errors*.

Phase overlap Even though the strict waterfall model envisages sharp transitions to occur from one phase to the next (see Figure 2.3), in practice the activities of different phases overlap (as shown in Figure 2.4) due to two main reasons:

- In spite of the best effort to detect errors in the same phase in which they are committed, some errors escape detection and are detected in a later phase.
- An important reason for phase overlap is that usually the work required to be carried out in a phase is divided among the team members. Some members may complete their part of the work earlier than other members.

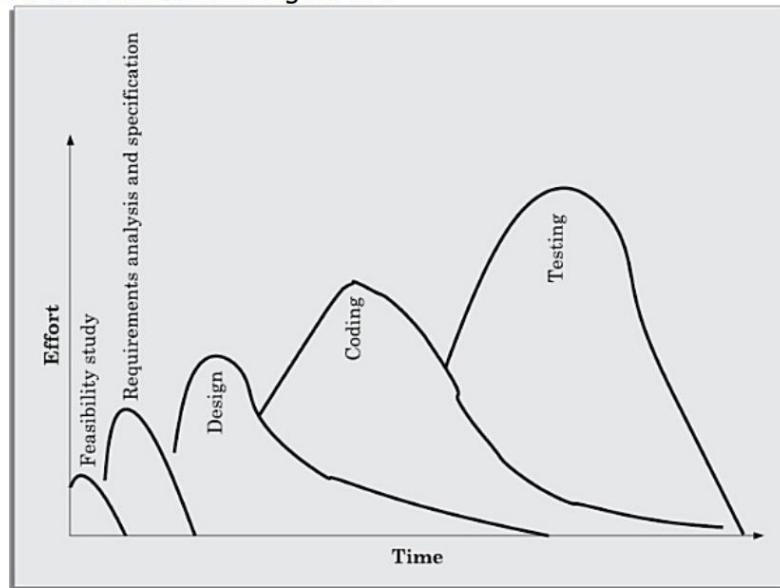


Figure 2.4: Distribution of effort for various phases in the iterative waterfall model.

Shortcomings of the iterative waterfall model

Difficult to accommodate change requests: A major problem with the waterfall model is that the requirements need to be frozen before the development starts.

Once requirements have been frozen, the waterfall model provides no scope for any modifications to the requirements.

Incremental delivery not supported: In the iterative waterfall model, the full software is completely developed and tested before it is delivered to the customer. There is no provision for any intermediate deliveries to occur.

Phase overlap not supported: For most real life projects, it becomes difficult to follow the rigid phase sequence prescribed by the waterfall model.

Error correction unduly expensive: In waterfall model, validation is delayed till the complete development of the software.

Limited customer interactions: This model supports very limited customer interactions. It is generally accepted that software developed in isolation from the customer is the cause of many problems.

Heavy weight: The waterfall model overemphasizes documentation. A significant portion of the time of the developers is spent in preparing documents, and revising them as changes occur over the life cycle.

No support for risk handling and code reuse: It becomes difficult to use the waterfall model in projects that are susceptible to various types of risks, or those involving significant reuse of existing development artifacts.

V-Model

A popular development process model, V-model is a variant of the waterfall model. As is the case with the waterfall model, this model gets its name from its visual appearance (see Figure 2.5). In this model verification and validation activities are carried out throughout the development life cycle, and therefore the chances bugs in the work products considerably reduce.

This model is therefore generally considered to be suitable for use in projects concerned with development of safety-critical software that are required to have high reliability.

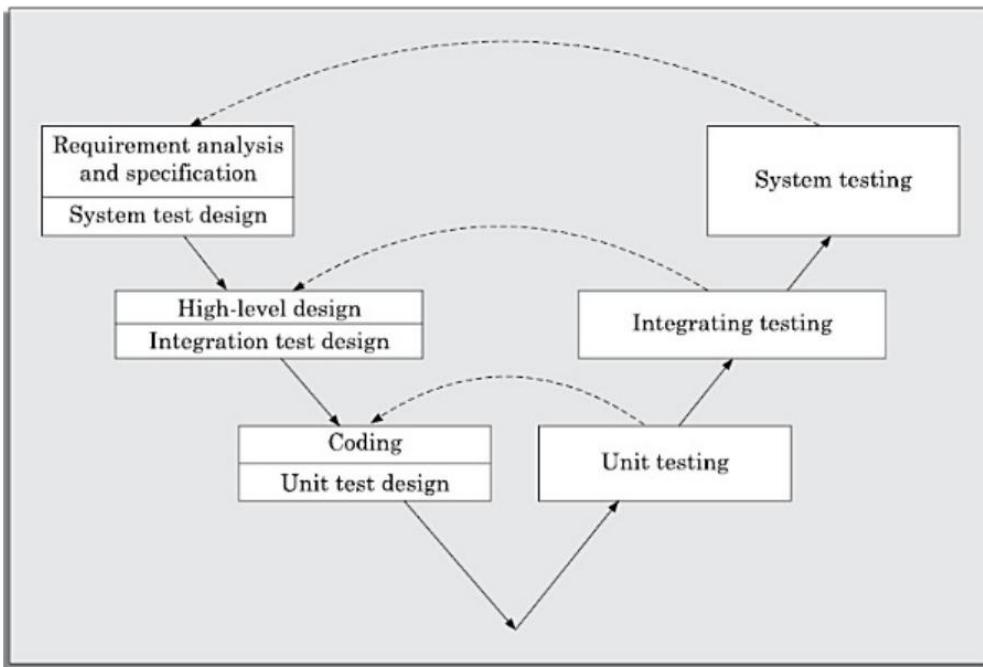


Figure 2.5: V-model.

The left half of the model comprises the development phases and the right half comprises the validation phases.

- In each development phase, along with the development of a work product, test case design and the plan for testing the work product are carried out, whereas the actual testing is carried out in the validation phase.
- In the validation phase, testing is carried out in three steps—unit, integration, and system testing. The purpose of these three different steps of testing during the validation phase is to detect defects that arise in the corresponding phases of software development—requirements analysis and specification, design, and coding respectively.

Advantages of V-model

The important advantages of the V-model over the iterative waterfall model are as following:

- In the V-model, much of the testing activities (test case design, test planning, etc.) are carried out in parallel with the development activities.
- Since test cases are designed when the schedule pressure has not built up, the quality of the test cases is usually better.
- The test team is reasonably kept occupied throughout the development cycle in contrast to the waterfall model where the testers are active only during the testing phase.
- In the V-model, the test team is associated with the project from the beginning. Therefore, they build up a good understanding of the development artifacts, and this in turn, helps them to carry out effective testing of the software.

Disadvantages of V-model

Being a derivative of the classical waterfall model, this model inherits most of the weaknesses of the waterfall model.

Prototyping Model

- The prototype model is also a popular life cycle model. The prototyping model can be considered to be an extension of the waterfall model.
- This model suggests building a working prototype of the system, before development of the actual software.

- A prototype is a toy and crude implementation of a system. It has limited functional capabilities, low reliability, or inefficient performance as compared to the actual software.
 - A prototype can be built very quickly by using several shortcuts.
 - The shortcuts usually involve developing inefficient, inaccurate, or dummy functions.

Life cycle activities of prototyping model

Prototype development:

- Prototype development starts with an initial requirements gathering phase.
 - A quick design is carried out and a prototype is built. The developed prototype is submitted to the customer for evaluation.
 - Based on the customer feedback, the requirements are refined and the prototype is suitably modified.
 - This cycle of obtaining customer feedback and modifying the prototype continues till the customer approves the prototype.

Iterative development:

- Once the customer approves the prototype, the actual software is developed using the iterative waterfall approach.
 - In spite of the availability of a working prototype, the SRS document is usually needed to be developed since the SRS document is invaluable for carrying out traceability analysis, verification, and test case design during later phases.

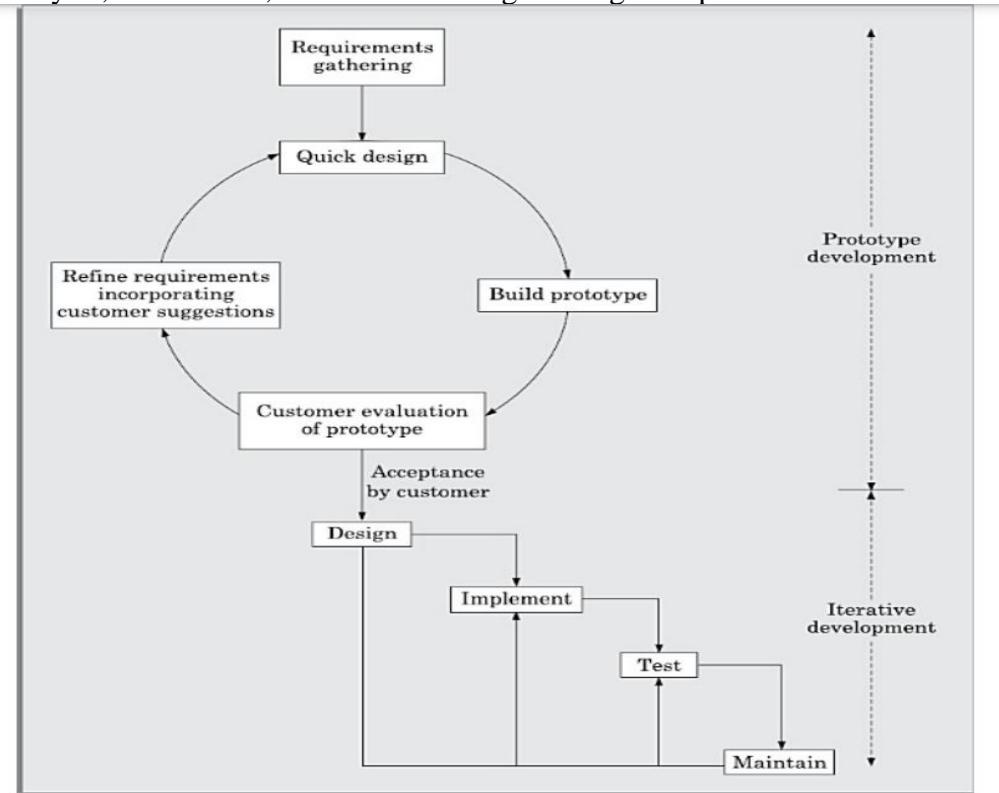


Figure 2.6: Prototyping model of software development.

Even though the construction of a throwaway prototype might involve incurring additional cost, for systems with unclear customer requirements and for systems with unresolved technical issues, the overall development cost usually turns out to be lower compared to an equivalent system developed using the iterative waterfall model.

Strengths of the prototyping model

This model is the most appropriate for projects that suffer from technical and requirements risks. A constructed prototype helps overcome these risks.

Weaknesses of the prototyping model

- The prototype model can increase the cost of development for projects that are routine development work and do not suffer from any significant risks.
- Even when a project is susceptible to risks, the prototyping model is effective only for those projects for which the risks can be identified upfront before the development starts.

Incremental Development Model

- This life cycle model is sometimes referred to as the successive versions model and sometimes as the incremental model.
- In this life cycle model, first a simple working system implementing only a few basic features is built and delivered to the customer.
- Over many successive iterations successive versions are implemented and delivered to the customer until the desired system is realized.

Life cycle activities of incremental development model

In the incremental life cycle model, the requirements of the software are first broken down into several modules or features that can be incrementally constructed and delivered. This has been pictorially depicted in Figure 2.7

- The development team first undertakes to develop the core features of the system. The core or basic features are those that do not need to invoke any services from the other features.
- On the other hand, non-core features need services from the core features. Once the initial core features are developed, these are refined into increasing levels of capability by adding new functionalities in successive versions. Each incremental version is usually developed using an iterative waterfall model of development.
- As each successive version of the software is constructed and delivered to the customer, the customer feedback is obtained on the delivered version and these feedbacks are incorporated in the next version.
- Each delivered version of the software incorporates additional features over the previous version and also refines the features that were already delivered to the customer.

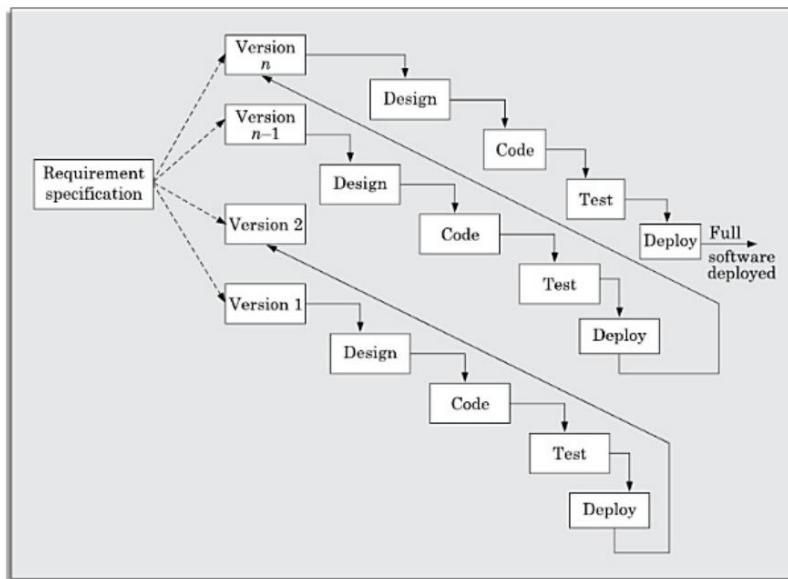


Figure 2.8: Incremental model of software development.

Advantages

The incremental development model offers several advantages. Two important ones are the following:

Error reduction: The core modules are used by the customer from the beginning and therefore these get tested thoroughly.

Incremental resource deployment: This model obviates the need for the customer to commit large resources at one go for development of the system

RAPID APPLICATION DEVELOPMENT (RAD)

The rapid application development (RAD) model was proposed in the early nineties in an attempt to overcome the rigidity of the waterfall model (and its derivatives) that makes it difficult to accommodate any change requests from the customer.

The major goals of the RAD model are as follows:

- To decrease the time taken and the cost incurred to develop software systems.
- To limit the costs of accommodating change requests.
- To reduce the communication gap between the customer and the developers.

Applicability of RAD Model The following are some of the characteristics of an application that indicate its suitability to RAD style of development:

1. Customised software
2. Non-critical software
3. Highly constrained project schedule
4. Large software

AGILE DEVELOPMENT MODELS

- The agile software development model was proposed in the mid-1990s to overcome the serious shortcomings of the waterfall model of development identified above.
- The agile model was primarily designed to help a project to adapt to change requests quickly.
- Thus, a major aim of the agile models is to facilitate quick project completion.

A few popular agile SDLC models are the following:

1. Crystal

2. Atern (formerly DSDM)
3. Feature-driven development
4. Scrum
5. Extreme programming (XP)
6. Lean development
7. Unified process

In the agile model, the requirements are decomposed into many small parts that can be incrementally developed.

The agile model adopts an iterative approach. Each incremental part is developed over an iteration. Each iteration is intended to be small and easily manageable and lasting for a couple of weeks only.

The agile model emphasises incremental release of working software as the primary measure of progress.

Principles of Agile process Model

The following important principles behind the agile model were publicised in the agile manifesto in 2001:

- Working software over comprehensive documentation.
- Frequent delivery of incremental versions of the software to the customer in intervals of few weeks.
- Requirement change requests from the customer are encouraged and are efficiently incorporated.
- Having competent team members and enhancing interactions among them is considered much more important than issues such as usage of sophisticated tools or strict adherence to a documented process.
- Continuous interaction with the customer is considered much more important rather than effective contract negotiation. A customer representative is required to be a part of the development team, thus facilitating close, daily co-operation between customers and developers.

Agile development projects usually deploy **pair programming**.

In pair programming, two programmers work together at one work station. One types in code while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Extreme Programming Model

- Extreme programming (XP) is an important process model under the agile umbrella and was proposed by Kent Beck in 1999.
- The name of this model reflects the fact that it recommends taking these best practices that have worked well in the past in program development projects to extreme levels.

In the following subsections, we mention some of the good practices that have been recognized in the extreme programming model and the suggested way to maximize their use:

Code review: It is good since it helps detect and correct problems most efficiently. It suggests pair programming as the way to achieve continuous review.

Testing: Testing code helps to remove bugs and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.

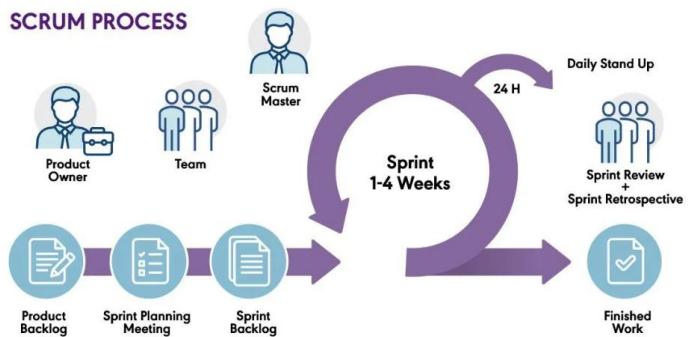
Incremental development: Incremental development is good, since it helps to get customer feedback, and extent of features delivered is a reliable indicator of progress.

Simplicity: Simplicity makes it easier to develop good quality code, as well as to test and debug it. Therefore, one should try to create the simplest code that makes the basic functionality being written to work.

Design: Since having a good quality design is important to develop a good quality solution, everybody should design daily

Scrum Model

- In the scrum model, a project is divided into small parts of work that can be incrementally developed and delivered over time boxes that are called sprints.
- The software therefore gets developed over a series of manageable chunks. Each sprint typically takes only a couple of weeks to complete.
- At the end of each sprint, stakeholders and team members meet to assess the progress made and the stakeholders suggest to the development team any changes needed to features that have already been developed and any overall improvements that they might feel necessary.



In the scrum model, the team members assume three fundamental roles— software owner, scrum master, and team member.

- The software owner is responsible for communicating the customers vision of the software to the development team.
- The scrum master acts as a liaison between the software owner and the team, thereby facilitating the development work.

SPIRAL MODEL

This model gets its name from the appearance of its diagrammatic representation that looks like a spiral with many loops (see Figure 2.10).

The exact number of loops of the spiral is not fixed and can vary from project to project.

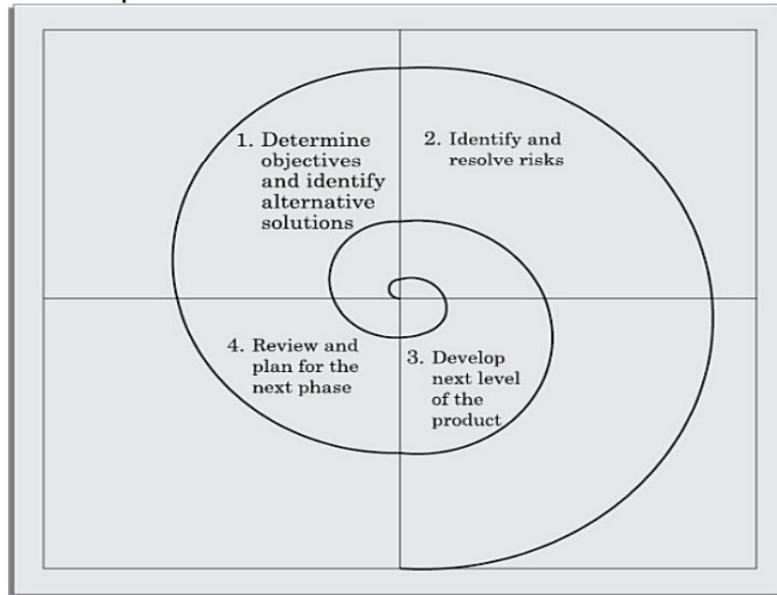


Figure 2.10: Spiral model of software development.

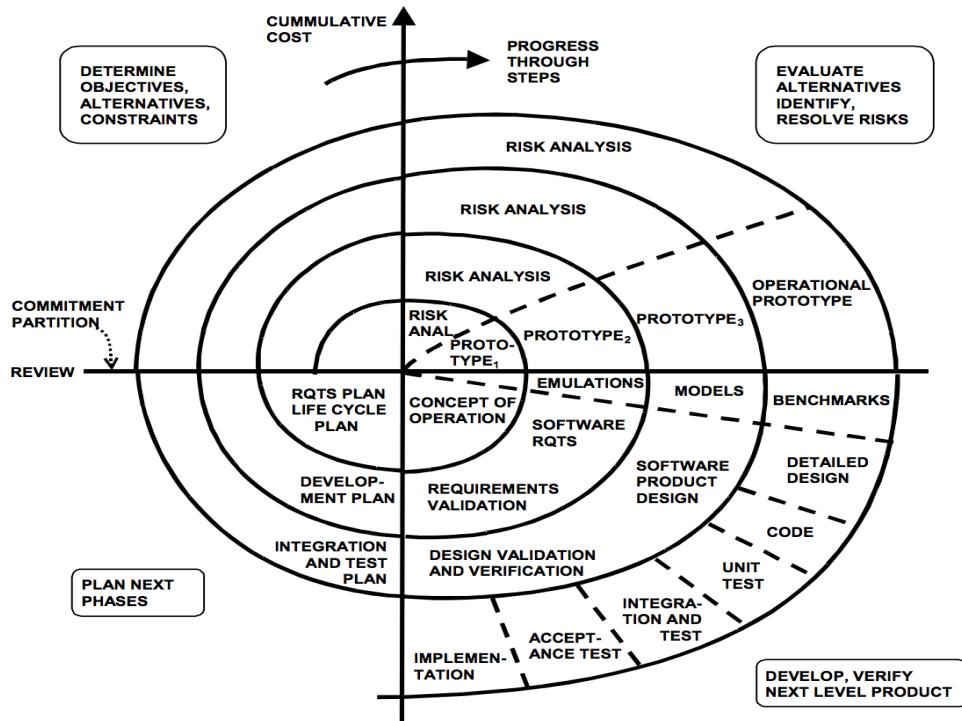
- The exact number of phases through which the product is developed can be varied by the project manager depending upon the project risks.
- The spiral model prototypes are built at the start of every phase. Each phase of the model is represented as a loop in its diagrammatic representation.
- Over each loop, one or more features of the product are elaborated and analyzed and the risks at that point of time are identified and are resolved through prototyping. Based on this, the identified features are implemented.

Quadrant 1: The objectives are investigated, elaborated, and analysed. Based on this, the risks involved in meeting the phase objectives are identified. In this quadrant, alternative solutions possible for the phase under consideration are proposed.

Quadrant 2: During the second quadrant, the alternative solutions are evaluated to select the best possible solution. To be able to do this, the solutions are evaluated by developing an appropriate prototype.

Quadrant 3: Activities during the third quadrant consist of developing and verifying the next level of the software. At the end of the third quadrant, the identified features have been implemented and the next version of the software is available.

Quadrant 4: Activities during the fourth quadrant concern reviewing the results of the stages traversed so far (i.e. the developed version of the software) with the customer and planning the next iteration of the spiral.



Advantages/pros and disadvantages/cons of the spiral model

There are a few disadvantages of the spiral model that

- restrict its use to only a few types of projects. To the developers of a project, the spiral model usually appears as a complex model to follow
- it is not very suitable for use in the development of outsourced projects, since the software risks need to be continually assessed as it is developed.

the advantages of the spiral model can outweigh its disadvantages.

For projects having many unknown risks that might show up as the development proceeds, the spiral model would be the most appropriate development model to follow.

In this regard, it is much more powerful than the prototyping model.

Prototyping model can meaningfully be used when all the risks associated with a project are known beforehand.

All these risks are resolved by building a prototype before the actual software development starts.