# Unit 2 Software Process Model

## Software Process

A software process is a set of related activities that leads to the production of a software system. There are many different types of software systems, and there is no universal software engineering method that is applicable to all of them. Consequently, there is no universally applicable software process. The process used in different companies depends on the type of software being developed, the requirements of the software customer, and the skills of the people writing the software.

Software process is defined as a framework for the task that is required to build high quality software. It is an approach that is taken as software is engineered. A structured set of activity required to develop a software system like: specification, design, validation and evolution are called **software process activities**. These process activities are organized differently in different development model. How these activities are carried out depends on the type of software, people, and organizational structure involved. Following are major **software process activities**:

1. **Software specification:** The functionality of the software and constraints on its operation must be defined.
2. **Software Design and Implementation:** The software to meet the specification must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing customer needs.

## Software specification

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. Requirements engineering is a particularly critical stage of the software process, as mistakes made at this stage inevitably lead to later problems in the system design and implementation.

Before the requirements engineering process starts, a company may carry out a feasibility or marketing study to assess whether or not there is a need or a market for the software and whether or not it is technically and financially realistic to develop the software required. Feasibility studies are short-term, relatively cheap studies that inform the decision of whether or not to go ahead with a more detailed analysis.

The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements. Requirements are usually presented at two levels of detail. End-users and customers need a high-level statement of the requirements; system developers need a more detailed system specification.

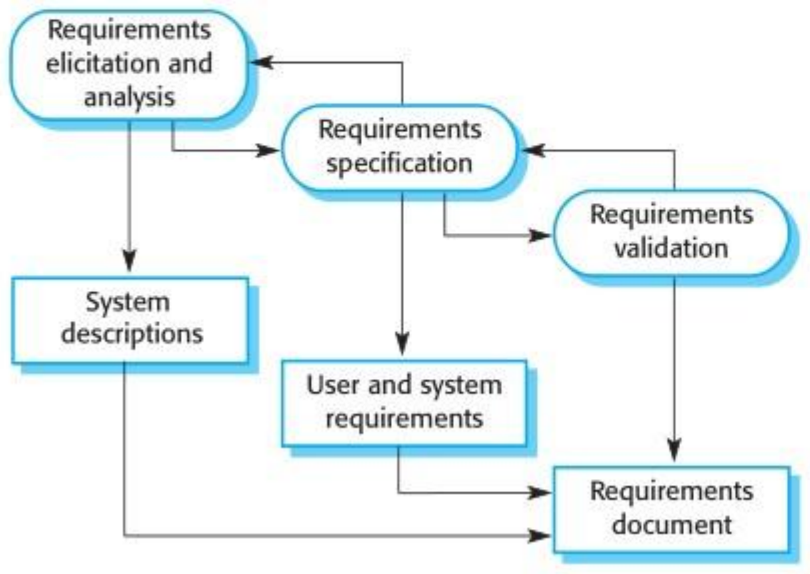

Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

**Fig: Requirement Specification Process**

**Software Design and Implementation**

Software design and implementation is the stage in the software engineering process at which an executable software system is developed. Software design and implementation activities are invariably inter-leaved.Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program. A software design is a description of the structure of the software to be implemented, the data which is part of system, the interfaces between system component, sometimes algorithm used, logical structure of database etc. The implementation of the software development is the process of converting system specification and design into an executable system. The specific design process activities include:

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design

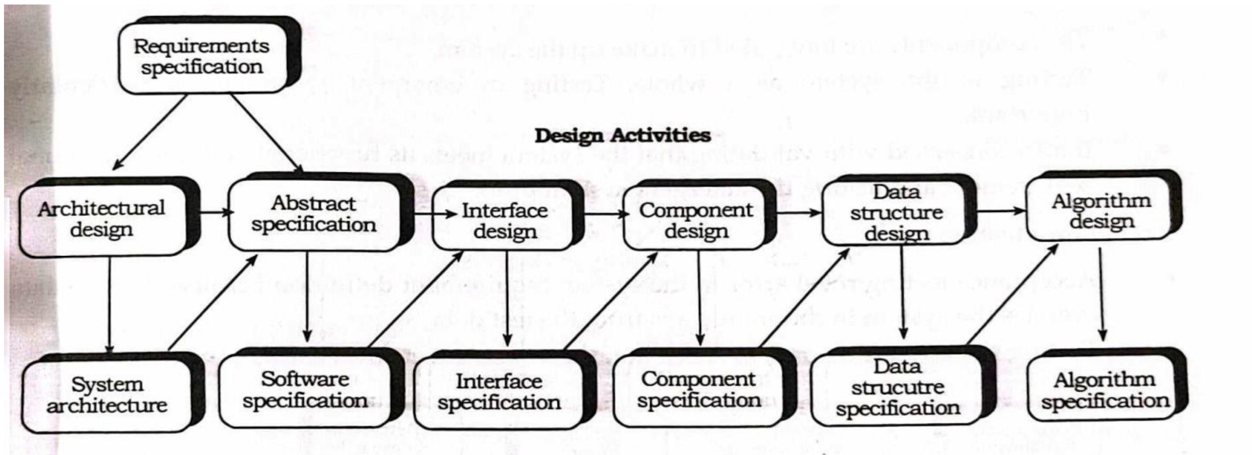


Fig: Design Products

## Software Validation

Software validation or, more generally, verification and validation (V & V) is intended to show that a system both conforms to its specification and meets the expectations of the system customer. Program testing, where the system is executed using simulated test data, is the principal validation technique. Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development.

However, most V & V time and effort is spent on program testing. Except for small programs, systems should not be tested as a single, monolithic unit. Figure shows a three-stage testing process in which system components are individually tested, and then the integrated system is tested.

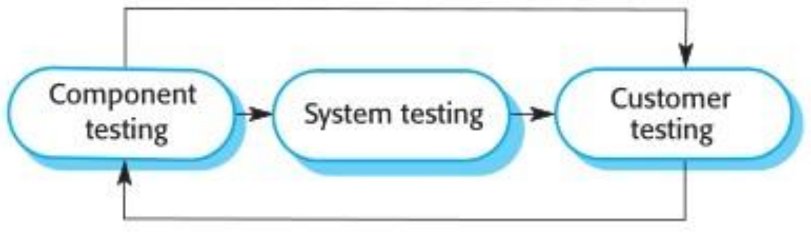


**Fig: System Testing Process**

For custom software, customer testing involves testing the system with real customer data. For products that are sold as applications, customer testing is sometimes called beta testing where selected users try out and comment on the software.

1. **Component testing** The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes or may be coherent groupings of these entities. Test automation tools, such as JUnit for Java, that can rerun tests when new versions of the component are created, are commonly used.
2. **System testing** System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems. It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties. For large systems, this may be a multistage process where components are integrated to form subsystems that are individually tested before these subsystems are integrated to form the final system.
3. **Customer testing** This is the final stage in the testing process before the system is accepted for operational use. The system is tested by the system customer (or potential customer) rather than with simulated test data. For custom-built software, customer testing may reveal errors and omissions in the system requirements definition, because the real data exercise the system in different ways from the test data. Customer testing may also reveal requirements problems where the system's facilities do not really meet

the users' needs or the system performance is unacceptable. For products, customer testing shows how well the software product meets the customer's needs.

**Software Evolution**

The flexibility of software is one of the main reasons why more and more software is being incorporated into large, complex systems. Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance). People think of software development as a creative activity in which a software system is developed from an initial concept through to a working system. However, they sometimes think of software maintenance as dull and uninteresting. They think that software maintenance is less interesting and challenging than original software development.

This distinction between development and maintenance is increasingly irrelevant. Very few software systems are completely new systems, and it makes much more sense to see development and maintenance as a continuum. Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

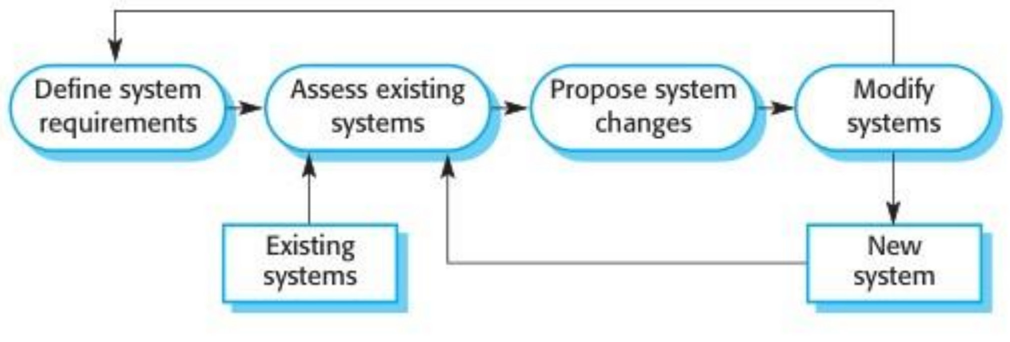


**Fig: System Evolution**

## Software Process Models

A software process model (sometimes called a Software Development Life Cycle or SDLC model) is a simplified representation of a software process. Software process model define a distinct set of activities, actions, tasks, milestones and work products that are required to engineer high quality software. They do provide a useful roadmap for software engineering work. It provides stability and control while developing software. A software process model is an abstract diagrammatic representation of a software process in a simplified form. It represents the order in which the activities of software development will be undertaken. Each model represents a process from a specific perspective. Depending on the size and purpose of an organization, the systems do differ in terms of their technological complexity and organizational problems they are meant to solve. As there are different kinds of systems, a **number of models** are in existence that can be used in the development of an information system. Some of them are explained below:

## Waterfall Model

This is the simplest software development life cycle model, which states that the phases are organized in a linear order. This model takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirement specification, design, implementation, testing and so on.

Because of cascade from one phase to another, this model is known as waterfall model or software life cycle or linear sequential model. In this model the result of each phase is one or more documents that are approved and following phase should not start until the previous phase has finished these stages and feed information to each other. The principal stages or waterfall model are depicted in the following figure.

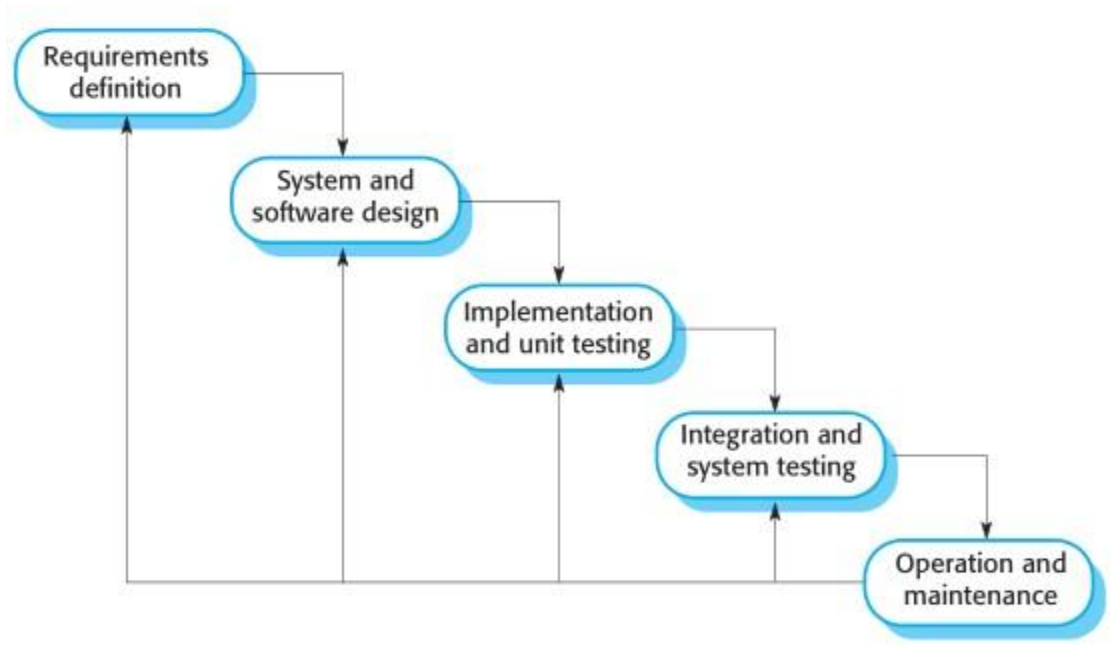

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Fig: Waterfall Model

The stages of the waterfall model directly reflect the fundamental software development activities:

1. **Requirements analysis and definition:** The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2. **System and software design** The systems design process allocates the requirements to either hardware or software systems. It establishes overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
3. **Implementation and unit testing** During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.
4. **Integration and system testing** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

5. **Operation and maintenance** Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered.

**Advantages of waterfall model**

- Linear structure of this model is easy to understand.
- Development progress is easily estimated.
- Easy to understand even by non-technical person, i.e. customers. .
- Each phase has well defined inputs and outputs
- Easy to use as software development proceeds.
- Each stage has well defined deliverables.
- Helps the project manager in proper planning of the project.

**Disadvantages of waterfall model**

- Difficulty of accommodating change after the process is underway.
- Real projects rarely follow the waterfall model because changes can cause confusion.
- It is often difficult for the user to state all requirements explicitly.
- It has unnecessary delays; sometimes it may lead to "blocking states".
- It has inflexible partitioning of the project into distinct stages
- This model is only appropriate when the requirements are well-understood

**Evolutionary Development Model/Incremental Development Model**

Incremental development is based on the idea of developing an initial implementation, getting feedback from users and others, and evolving the software through several versions until the required system has been developed. Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.

Incremental development in some form is now the most common approach for the development of application systems and software products. This approach can be either plan-driven, agile or, more usually, a mixture of these approaches. In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified, but the development of later increments depends on progress and customer priorities. Incremental software development, which is a fundamental part of agile development methods, is better than a waterfall approach for systems whose requirements are likely to change during the development process. This is the case for most business systems and software products. Incremental development reflects the way that we solve problems. We rarely work out a complete problem solution in advance but move toward a solution in a series of steps, backtracking when we realize that we have made a mistake. By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.

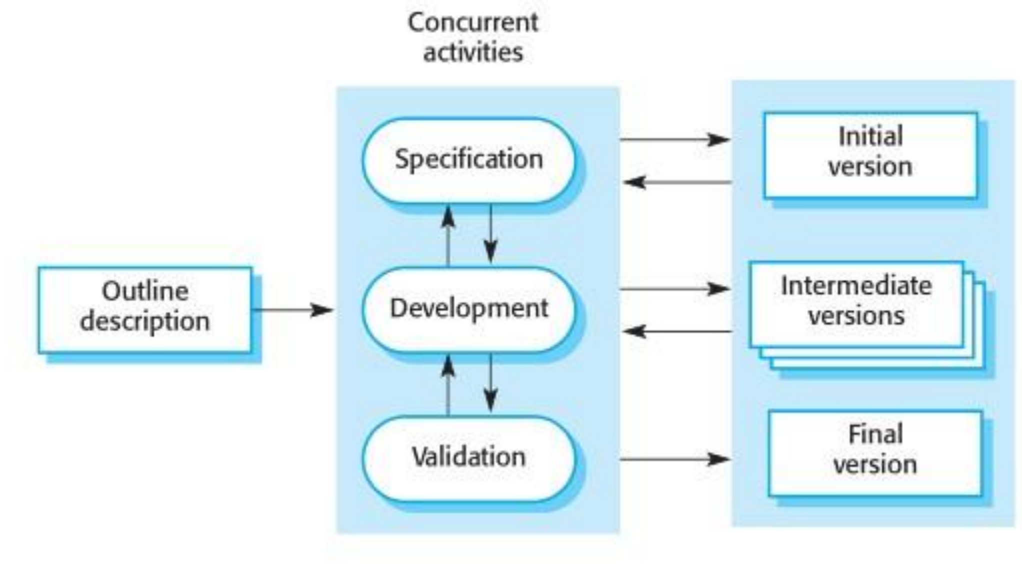


**Fig: Incremental Development Model**

**Advantages**

- Deals constantly with changes Provides quickly an initial version of the system
- Involves all development teams

**Disadvantages**

- Quick fixes may be involved,
- The system's structure can be corrupted by continuous change,
- Special tools and techniques may be necessary.

**Prototyping Model**

In traditional waterfall model the intermediate changes in user's requirements cannot be accommodated once the system development process begins. So, an alternative method to traditional method has been developed, called prototyping model is based on the assumption that it is difficult to know all the requirement of user in advance. A prototyping is an initial version of software that is used to demonstrate the concept, tryout design options and find out more about problem domain and its possible solutions.

The basic idea in Prototype model is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. It is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. In many instances the user only has general view of what is expected from the software product. In such a scenario where there is absence of detailed information regarding the input to the system, the processing needs and the output requirement, the prototyping model may be employed. This model reflects an attempt to increase the flexibility of the development process by allowing the client to interact and experiment with a working representation of the product.

This type of System development method is employed when it is very difficult to obtain exact requirements from the customer (unlike waterfall model, where requirements are clear). While making the model, user keeps giving feedbacks from time to time and based on it, a prototype is made. Completely built sample model is shown to user and based on his feedback; the SRS (System Requirements Specifications) document is prepared. After completion of this, a more accurate SRS is prepared, and now development work can start using Waterfall Model.

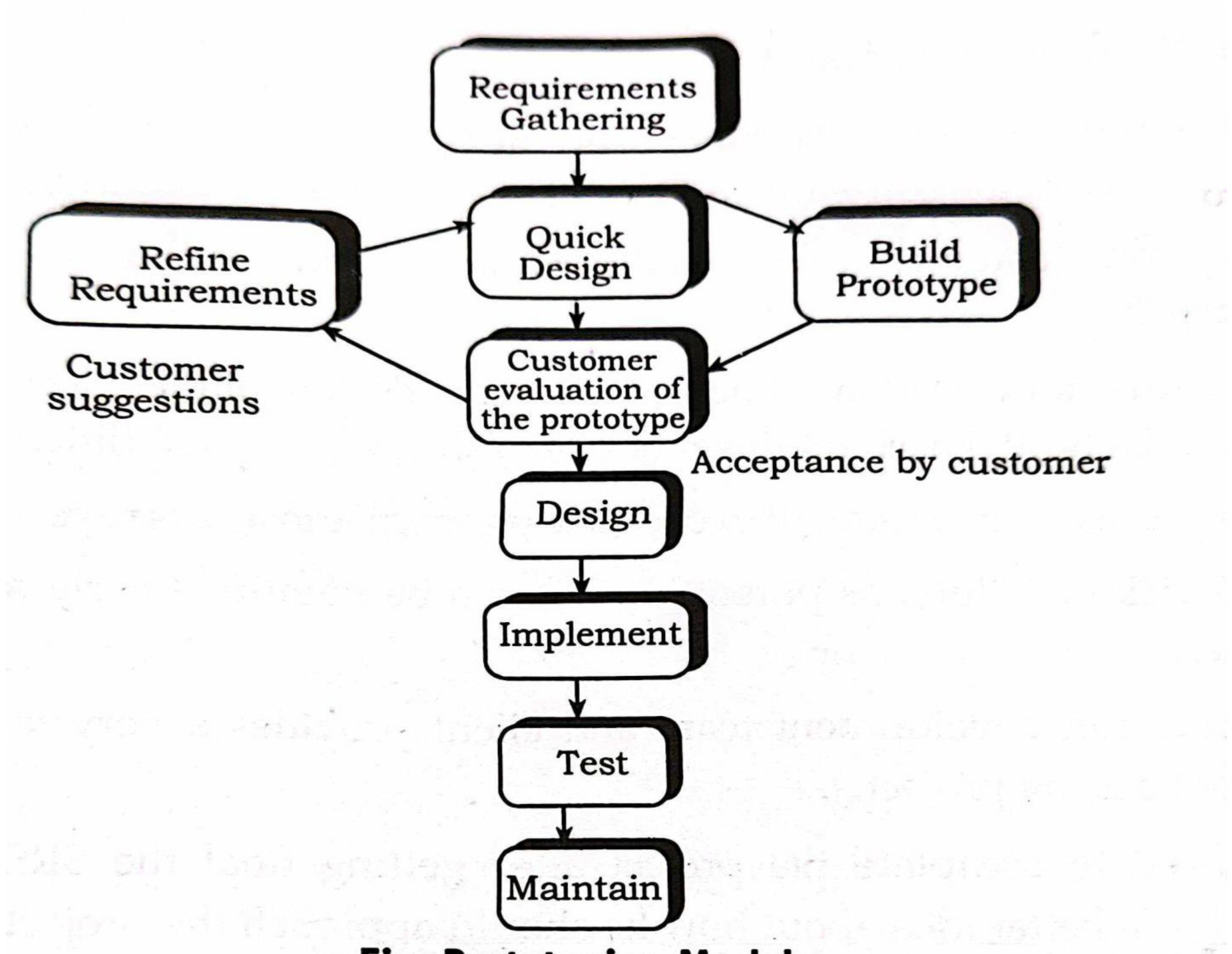


Fig: Prototyping Model

**The prototyping model consists of following steps:**

1. **Determine the Requirements**: This is the first phase of prototyping model. In this, the system's service, constraints and goals are established by consultation with system users.
2. **Quick Design**: On the basis of known requirement, rough design of system is drawn for discussion and decision
3. **Build/create Prototype:** In this phase, the information from the design is rolled into prototype. Prototype is the blueprint of the system.
4. **Use and Evaluate Prototyping:** In this phase, the prototype is presented to the user of the system foe use and review. User comments and suggestions are collected and areas of refinements are determined.

5. **Refine the Prototype:** According to suggestions from the user, the system developer revises and enhances the prototype to make it more effective and efficient to me customer requirements.
6. **Design:** Once the prototype is refined and accepted by the users, actual system is designed to satisfy customer.
7. **Implementation:** During this stage, the software design is realized as a set of programs and program units
8. **Testing:** Once the program modules are ready, each of the program modules is tested independently as per the specifications of the users and debugged.
9. **Operation and Maintenance**: In this stage, the system is installed and put into practical use. Maintenance involves correcting and improving the implementation of system units and enhancing the system's services as new requirements are discovered.

**Advantages of Prototyping Model**

- When prototype is shown to the user, he/she gets a proper clarity and feet of the functionality of the software and he can suggest changes and modifications.
- Feedbacks from customer are received periodically and the changes don't come as a last minute surprise
- When client is not confident about the developer's capabilities, he/she asks for a small prototype to be built. Based on this model, he/she judges capabilities of developer.
- Sometimes it helps to demonstrate the concept to prospective investors to get funding for project

**Disadvantages of Prototyping Model**

- Prototyping is usually done at the cost of the developer. So it should be done using minimal resources.
- Once we get proper requirements from client after showing prototype model, it may be of no use. That is why; sometimes we refer to the prototype as "Throw-away" prototype.
- Too much involvement of client is not always preferred by the developer.
- Too many changes can disturb the rhythm of the development team.

**Spiral Model**

The spiral model, also known as spiral lifecycle model or Boehm's model was introduced by Barry Boehm in 1988. It is a system development model based on process iteration. This model of development combines the features of prototyping model and the waterfall model in order to eliminate almost every possible/ known risk factor from it. This model follows a spiral process as the development of the software takes place. In this model the development of each modified version of the system prototype is carefully designed using the steps of waterfall model. Starting from the center of spiral process, and moving outward in the spiral route, each iteration helps to build a more complete version of the system in a progressive manner. The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has following major phases:

1. Customer Communication
2. Planning
3. Risk Analysis
4. Engineering
5. Construction and Release
6. Customer Evaluation.

These phases can be depicted in the following figure:

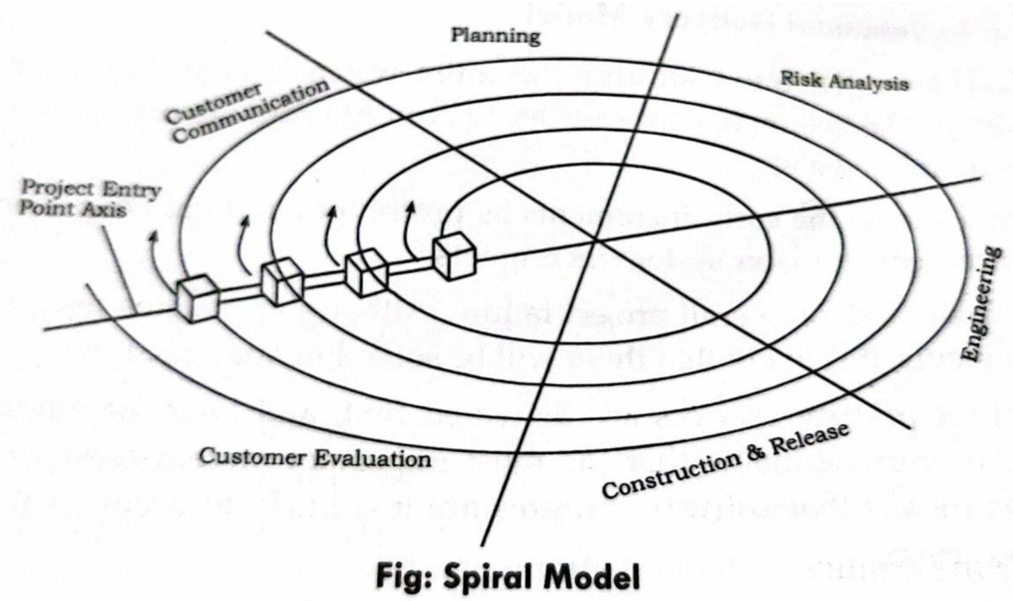


Fig: Spiral Model

1. **Customer Communication**: In this step, effective communication is established between developer and customer to identify the customer requirements
2. **Planning:** In this phase, the objectives, alternatives and constraints of the projects are determined and are documented. The objectives and other specifications are fixed in order to decide which strategies to follow during project life cycle.
3. **Risk analysis**: This is the most important phase of spiral model, in this phase all possible alternatives, which can help in developing a cost effective project are analyzed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risk in the project development. A prototype is produced at the end of the risk analysis phase to proceed with available data.
4. **Engineering:** In this phase, the actual development of the project is carried out. The output of this phase is passed through all the phases iteratively in order to obtain improvements in same.
5. **Construction and release**: Tasks required constructing, testing, installing, and providing user support (e.g., documentation and training).
6. **Customer evaluation**: In this phase, developed product is passed onto the customer in order to receive customer comments and suggestions which can help in identifying and resolving potential in the developed software. This phase is much similar to testing phase. This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

**Advantages of Spiral Model**

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

**Disadvantages of Spiral Model**

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

**Coping with Change**

Change is inevitable in all large software projects. The system requirements change as businesses respond to external pressures, competition, and changed management priorities. As new technologies become available, new approaches to design and implementation become possible. Therefore whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework. For example, if the relationships between the requirements in a system have been analyzed and new requirements are then identified, some or all of the requirements analysis has to be repeated. It may then be necessary to redesign the system to deliver the new requirements, change any programs that have been developed, and retest the system.

Two related approaches may be used to reduce the costs of rework:

1. **Change anticipation**, where the software process includes activities that can anticipate or predict possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers. They can experiment with the prototype and refine their requirements before committing to high software production costs.
2. **Change tolerance**, where the process and software are designed so that changes can be easily made to the system. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have to be altered to incorporate the change.

**Two ways of coping with change and changing system requirements:**

1. **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This is a method of change anticipation as it allows users to experiment with the system before

delivery and so refine their requirements. The number of requirements change proposals made after delivery is therefore likely to be reduced.

2. **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

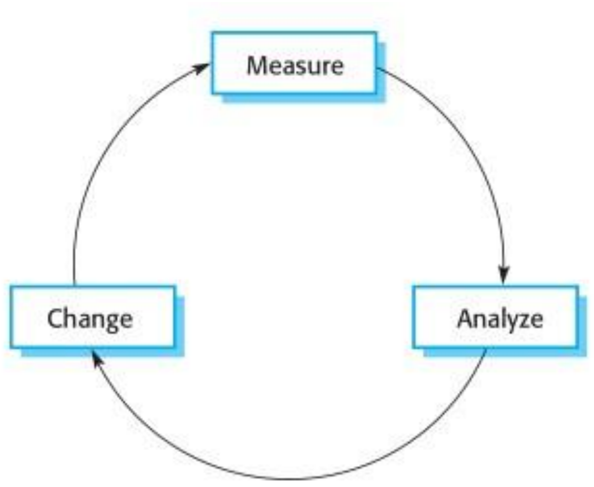
## Process Improvement

Nowadays, there is a constant demand from industry for cheaper, better software, which has to be delivered to ever-tighter deadlines. Consequently, many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs, or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Two quite different approaches to process improvement and change are used:

1. **The process maturity approach**, which has focused on improving process and project management and introducing good software engineering practice into an organization. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes. The primary goals of this approach are improved product quality and process predictability.
2. **The agile approach,** which has focused on iterative development and the reduction of overheads in the software process. The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements. The improvement philosophy here is that the best processes are those with the lowest overheads and agile approaches can achieve this.

The general process improvement process underlying the process maturity approach is a cyclical process, as shown in figure.

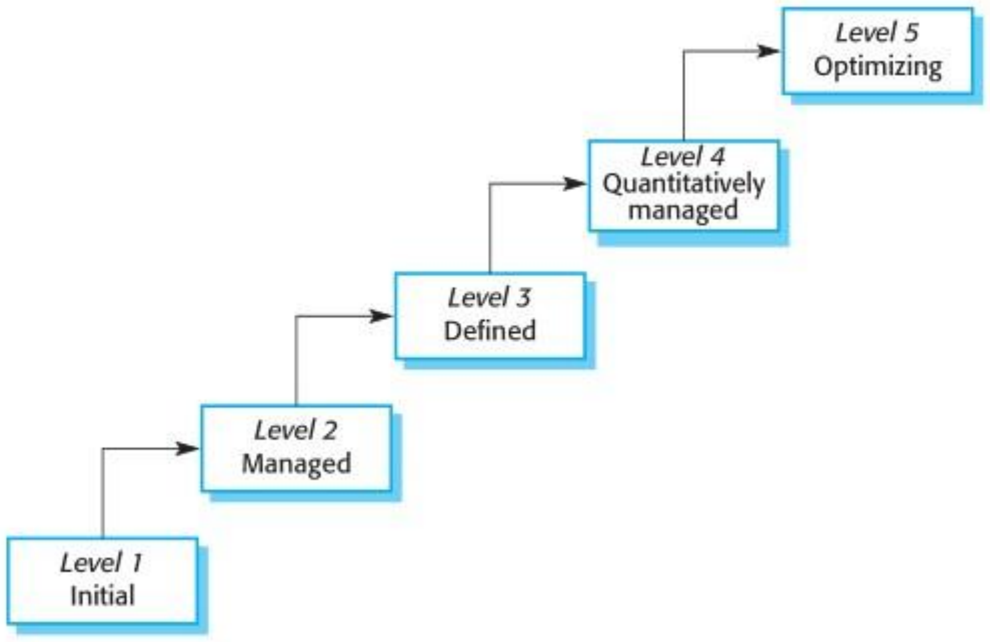


The stages in this process are:

1. **Process measurement** You measure one or more attributes of the software process or product. These measurements form a baseline that helps you decide if process improvements have been effective. As you introduce improvements, you re-measure the same attributes, which will hopefully have improved in some way.
2. **Process analysis** The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed during this stage. The analysis may be focused by considering process characteristics such as rapidity and robustness.
3. **Process change** Process changes are proposed to address some of the identified process weaknesses. These are introduced, and the cycle resumes collecting data about the effectiveness of the changes.

Process improvement is a long-term activity, so each of the stages in the improvement process may last several months. It is also a continuous activity as, whatever new processes are introduced, the business environment will change and the new processes will themselves have to evolve to take these changes into account.

The notion of **process maturity** was introduced in the late 1980s when the Software Engineering Institute (SEI) proposed their model of process capability maturity (Humphrey 1988). The maturity of a software company's processes reflects the process management, measurement, and use of good software engineering practices in the company. This idea was introduced so that the U.S. Department of Defense could assess the software engineering capability of defense contractors, with a view to limiting contracts to those contractors who had reached a required level of process maturity. Five levels of process maturity were proposed as shown in figure.



The levels in the process maturity model are:

1. **Initial** The goals associated with the process area are satisfied, and for all processes the scope of the work to be performed is explicitly set out and communicated to the team members.
2. **Managed** At this level, the goals associated with the process area are met, and organizational policies are in place that defines when each process should be used. There must be documented project plans that define the project goals. Resource management and process monitoring procedures must be in place across the institution.
3. **Defined** This level focuses on organizational standardization and deployment of processes. Each project has a managed process that is adapted to the project requirements from a defined set of organizational processes. Process assets and process measurements must be collected and used for future process improvements.

4. **Quantitatively managed** At this level, there is an organizational responsibility to use statistical and other quantitative methods to control sub processes. That is, collected process and product measurements must be used in process management.
5. **Optimizing** At this highest level, the organization must use the process and product measurements to drive process improvement. Trends must be analyzed and the processes adapted to changing business needs.

The work on process maturity levels has had a major impact on the software industry. It focused attention on the software engineering processes and practices that were used and led to significant improvements in software engineering capability. However, there is too much overhead in formal process improvement for small companies, and maturity estimation with agile processes is difficult. Consequently, only large software companies now use this maturity-focused approach to **software process improvement**.

## Agile Software Development

Agile software development refers to methods and practices that provide value quickly, efficiently, and consistently to customers as it relates to the software development lifecycle (SDLC). The ability to build and react to change is called **Agile.** It is a method for coping and, ultimately, succeeding in an uncertain and turbulent development environment. Within the Agile software development model, self-organizing and cross-functional teams work together to build and deploy solutions. Some of the popular Agile methodologies include Scrum, Kanban, and Lean.

The Agile software development paradigm is a software development methodology comprising practices and approaches that thrive on iterative and incremental software development. In this development methodology, the requirements – as well as the solutions – evolve through collaboration amongst self-organizing, cross-functional programmer teams that may not or may not be collocated or remote.

The Agile approach fosters adaptability, evolutionary development, and delivery, as well as a time-bound, iterative approach and quick response to change. Agile promotes adaptive planning, evolutionary growth, early delivery, and continual improvement.

## The Agile Lifecycle

A typical Agile lifecycle consists of the following steps:

- **Project Planning** – This helps your team understand the goals, the value to be delivered to the stakeholders, and define the project scope.
- **Product Roadmap** – This helps to define a breakdown of all the features that are needed as part of the final deliverable.
- **Release Planning** – This helps plan the future releases and revisit and reevaluate the release plans before a sprint starts.
- **Sprint Planning** – This helps plan how the tasks in a sprint should be accomplished, by whom and the time it would take to complete those tasks.
- **Daily Scrum Meetings** – These meetings are usually short and help the team know the tasks to be accomplished on a particular day, the roadblocks (if any) and assess if any changes are required.
- **Sprint Review / Retrospective Meetings** – These meetings are usually held after every sprint to discuss what went well in the sprint and what did not or what could have been done better.