Chapter

3

SOFTWARE DEVELOPMENT PROCESS MODELS

3.1 INTRODUCTION

The objective of this chapter is to introduce you to the idea of a Software process—a coherent set of activities for Software production. When you have read this chapter you will:

- understand the concepts of Software processes and Software process models;
- have been introduced to three generic Software process models and when they might be used;
- know about the fundamental process activities of Software requirements engineering, Software development, testing, and evolution;
- understand why processes should be organized to cope with changes in the Software requirements and design;
- understand how the Rational Unified Process integrates good Software engineering practice to create adaptable Software processes.

3.2 WHAT IS SOFTWARE PRODUCT?

Computer program with a set of documentation (Software design and structure and user documentation), configuration data and all additional sources of information directly related to the program

There are 2 types of Software product:

1. **Generic product.** Stand-alone Softwares, sold on open market to everyone. Examples – desktop operating Software, office packs or graphical editors.
2. **Customized** (bespoke) products. Developed for particular customers with special features. Examples – special Softwares for traffic control or airbus flight operations.

From developers' stand of view, one of the main differences of those types is that for customized case the specifications for a Software product is done mostly by the organization which buying it. For generic product development, specifications are created by the same development organization.

Nowadays, there is no straight line between those types as there was several years ago.

Software engineers, as any other engineers, make things work by applying theories, methods and tools (created, mostly, by computer scientists, but when it comes to practical problems, there's often need to change the solution and develop your own way, still, based on some theory). Software engineering is not all about coding programs, there are much more problems to solve during the work on project, for example project management, scheduling of work, testing and choosing the right way to go.

Software engineering is concerned with all aspects of the development and evolution of complex Softwares. Software engineering is concerned with hardware development, policy and process design.

Software process consists of four fundamental activities:

1. Software specification where engineers or/and customers define what the product should do and how should it operate.
2. Software development is designing and actual coding.
3. Software validation is generally testing. It is important to check if the Software is designed and implemented correctly.
4. Software evolution is modifying the Software according to new needs of customer(s).

3.3 WHAT ARE SOFTWARE PROCESS MODELS ?

A Process Model describes the sequence of phases for the entire lifetime of a product. Therefore it is sometimes also called Product Life Cycle. This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use.

Usually there are three main phases:

- ❒ Concept phase
- ❒ Implementation phase
- ❒ Maintenance phase

Each of these main phases usually has some sub-phases, like a requirements engineering phase, a design phase, a build phase and a testing phase. The sub-phases may occur in more than one main phase each of them with a specific peculiarity depending on the main phase.

Besides the phases a Process Model shall also define at least:

- ❒ The **activities** that have to be carried out in each of the sub-phases, including the sequence in which these activities have to be carried out.
- ❒ The **roles** of the executors that have to carry out the activities, including a description of their responsibilities and required skills.
- ❒ The **work products** that have to be established or updated in each of the activities. Besides the final product there are usually several other items that have to be generated during the development of a product. These are for example requirements and design document, test specifications and test reports, etc.

Therefore, a Process Model provides a fixed framework that guides a project in:

- Development of the product
- Planning and organizing the project
- Tracking and running the project

The development models are the various processes or methodologies that are being selected for the development of the project depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out.

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

There are various Software development models or methodologies. They are as follows:

1. **Classical Waterfall Model**
2. **The Waterfall Model**
3. **Iterative Model**
4. **Spiral Model**
5. **The V-Model**
6. **Big Bang Model**
7. **Agile Model**
8. **Rad Model**
9. **Prototyping Model**
10. **Build and Fix Model**
11. **Incremental Model**
12. **Synchronize-and-Stabilize Model**
13. **Evolutionary Model**
14. **Generation Techniques (FGT)**

3.3.1 Classical Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop Software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it cannot be used in actual Software development projects. Thus, this model can be considered to be a *theoretical way of developing Software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other

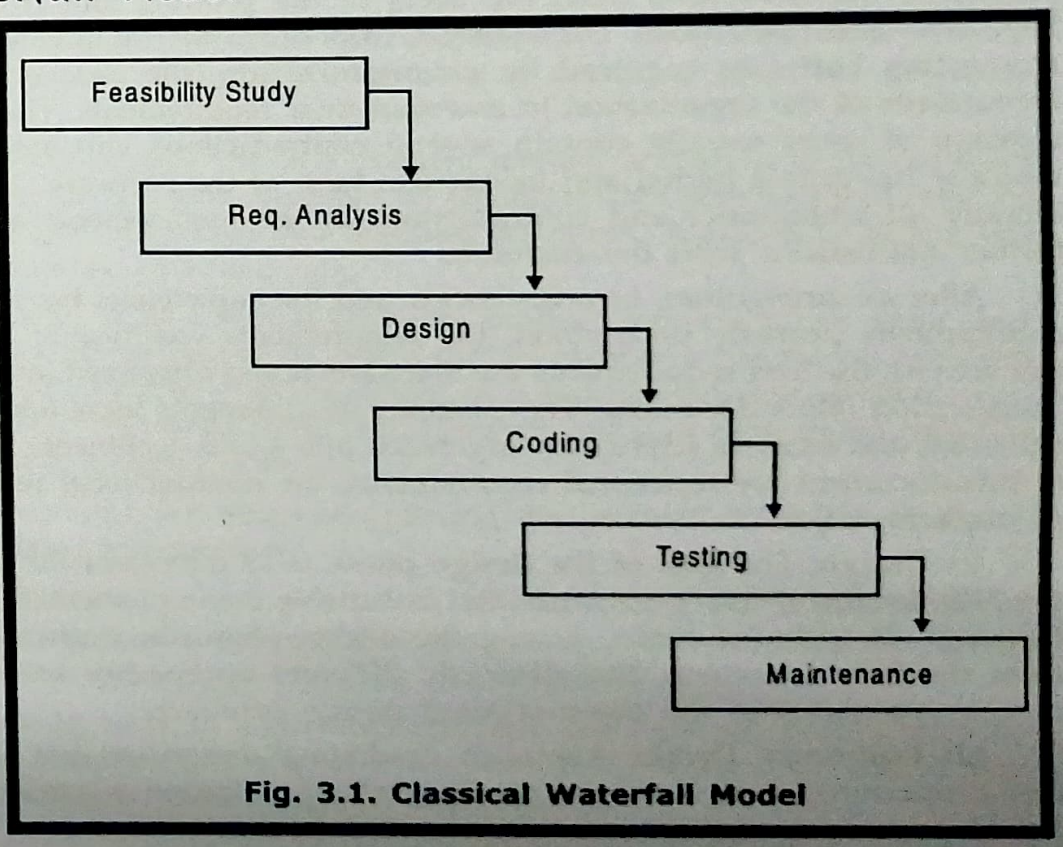

Fig. 3.1. Classical Waterfall Model

life cycle models it is necessary to learn the classical waterfall model. Classical waterfall model divides the life cycle into the following phases as shown in fig. 4.1.

(a) Feasibility Study: The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the Software and output data to be produced by the Software. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the Software.

- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.
- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

(b) 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. This phase consists of two distinct activities, namely.

- Requirements gathering and analysis
- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting Software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the Software. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer.

After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are Softwareatically organized into a Software Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.

(c) 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. In technical terms, during the design phase the Software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

(d) Traditional Design Approach: Traditional design consists of two different activities: first a structured analysis of the requirements specification is carried out where the detailed

structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the Software design.

- **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.

(e) Coding and Unit Testing: The purpose of the coding phase (sometimes called the implementation phase) of Software development is to translate the Software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

(f) Integration and Software Testing: Integration of different modules is undertaken once they have been coded and unit tested. During the integration and Software testing phase, the modules are integrated in a planned manner. The different modules making up a Software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated Software is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, Software testing is carried out. The goal of Software testing is to ensure that the developed Software conforms to its requirements laid out in the SRS document. Software testing usually consists of three different kinds of testing activities:

- **a–testing:** It is the Software testing performed by the development team.
- **b–testing:** It is the Software testing performed by a friendly set of customers.
- **Acceptance testing:** It is the Software testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Software testing is normally carried out in a planned manner according to the Software test plan document. The Software test plan identifies all testing-related activities that must be performed specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

(g) Maintenance: Maintenance of a typical Software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical Software product to its maintenance effort is roughly in the 40:60 ratios. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the Software, and enhancing the functionalities of the Software according to the customer's requirements. This is called perfective maintenance.
- Porting the Software to work in a new environment. For example, porting may be

life cycle models it is necessary to learn the classical waterfall model. Classical waterfall model divides the life cycle into the following phases as shown in fig. 4.1.

(a) Feasibility Study: The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the Software and output data to be produced by the Software. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the Software.

- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.
- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

(b) 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. This phase consists of two distinct activities, namely.

- Requirements gathering and analysis
- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting Software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the Software. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer.

After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are Softwareatically organized into a Software Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.

(c) 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. In technical terms, during the design phase the Software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

(d) Traditional Design Approach: Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed

structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the Software design.

- **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.

(e) Coding and Unit Testing: The purpose of the coding phase (sometimes called the implementation phase) of Software development is to translate the Software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

(f) Integration and Software Testing: Integration of different modules is undertaken once they have been coded and unit tested. During the integration and Software testing phase, the modules are integrated in a planned manner. The different modules making up a Software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated Software is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, Software testing is carried out. The goal of Software testing is to ensure that the developed Software conforms to its requirements laid out in the SRS document. Software testing usually consists of three different kinds of testing activities:

- **a–testing:** It is the Software testing performed by the development team.
- **b–testing:** It is the Software testing performed by a friendly set of customers.
- **Acceptance testing:** It is the Software testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Software testing is normally carried out in a planned manner according to the Software test plan document. The Software test plan identifies all testing-related activities that must be performed specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

(g) Maintenance: Maintenance of a typical Software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical Software product to its maintenance effort is roughly in the 40:60 ratios. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the Software, and enhancing the functionalities of the Software according to the customer's requirements. This is called perfective maintenance.
- Porting the Software to work in a new environment. For example, porting may be required to get the Software to work on a new computer platform or with a new operating Software. This is called adaptive maintenance.

3.3.2. Shortcomings of the Classical Waterfall Model

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical Software development work, it is not possible to strictly follow the classical waterfall model.

3.3.3 The Waterfall Model

The Waterfall Model was 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. 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; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap.

3.3.3.1. 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 Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.

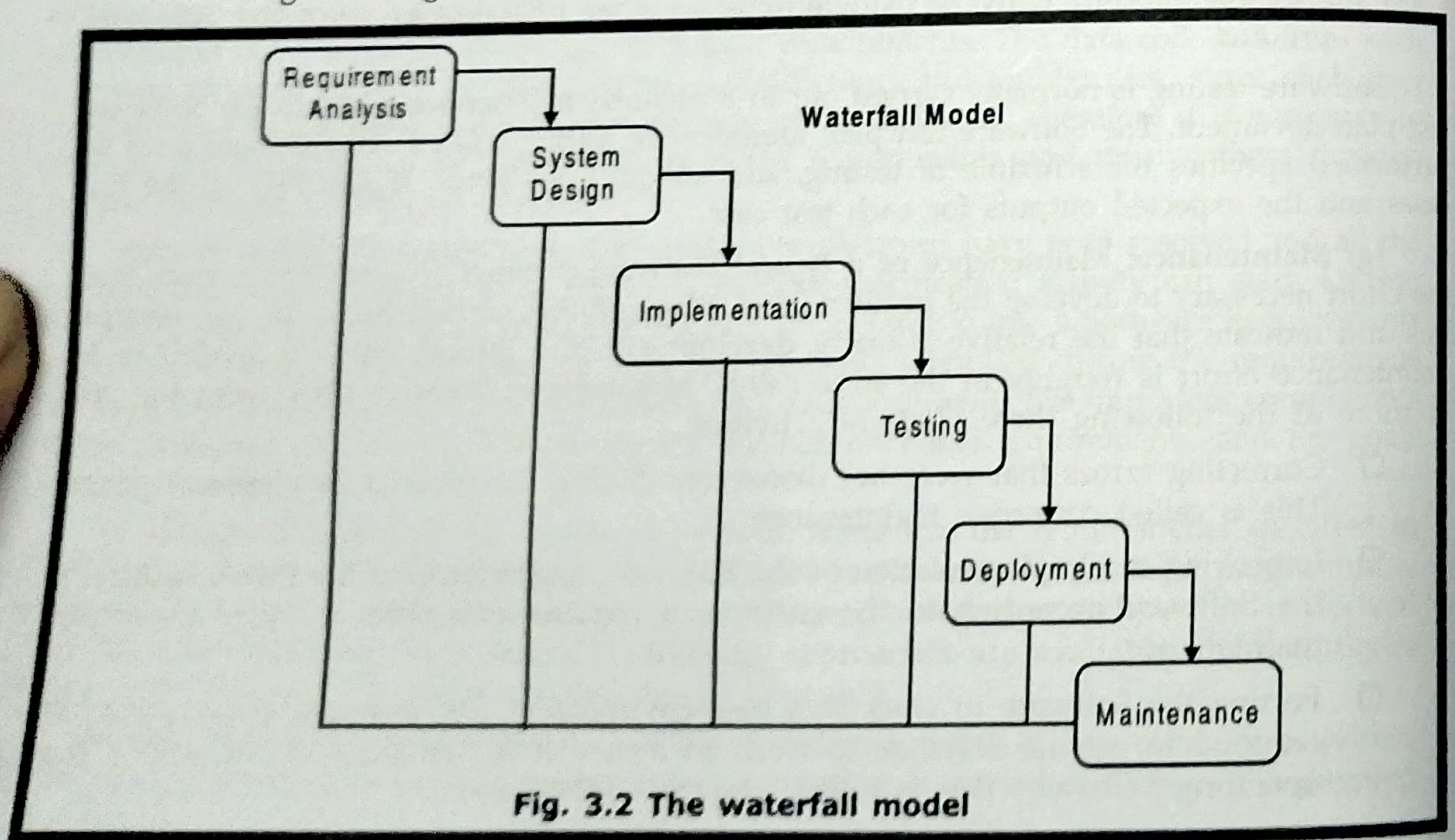

Fig. 3.2 The waterfall model

The sequential phases in Waterfall model are:

- **Requirement Gathering and analysis:** All possible requirements of the Software to be developed are captured in this phase and documented in a requirement specification doc.
- **Software Design:** The requirement specifications from first phase are studied in this phase and Software design is prepared. Software Design helps in specifying hardware and Software requirements and also helps in defining overall Software architecture.
- **Implementation:** With inputs from Software design, the Software 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 Software after testing of each unit. Post integration the entire Software is tested for any faults and failures.
- **Deployment of Software:** 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

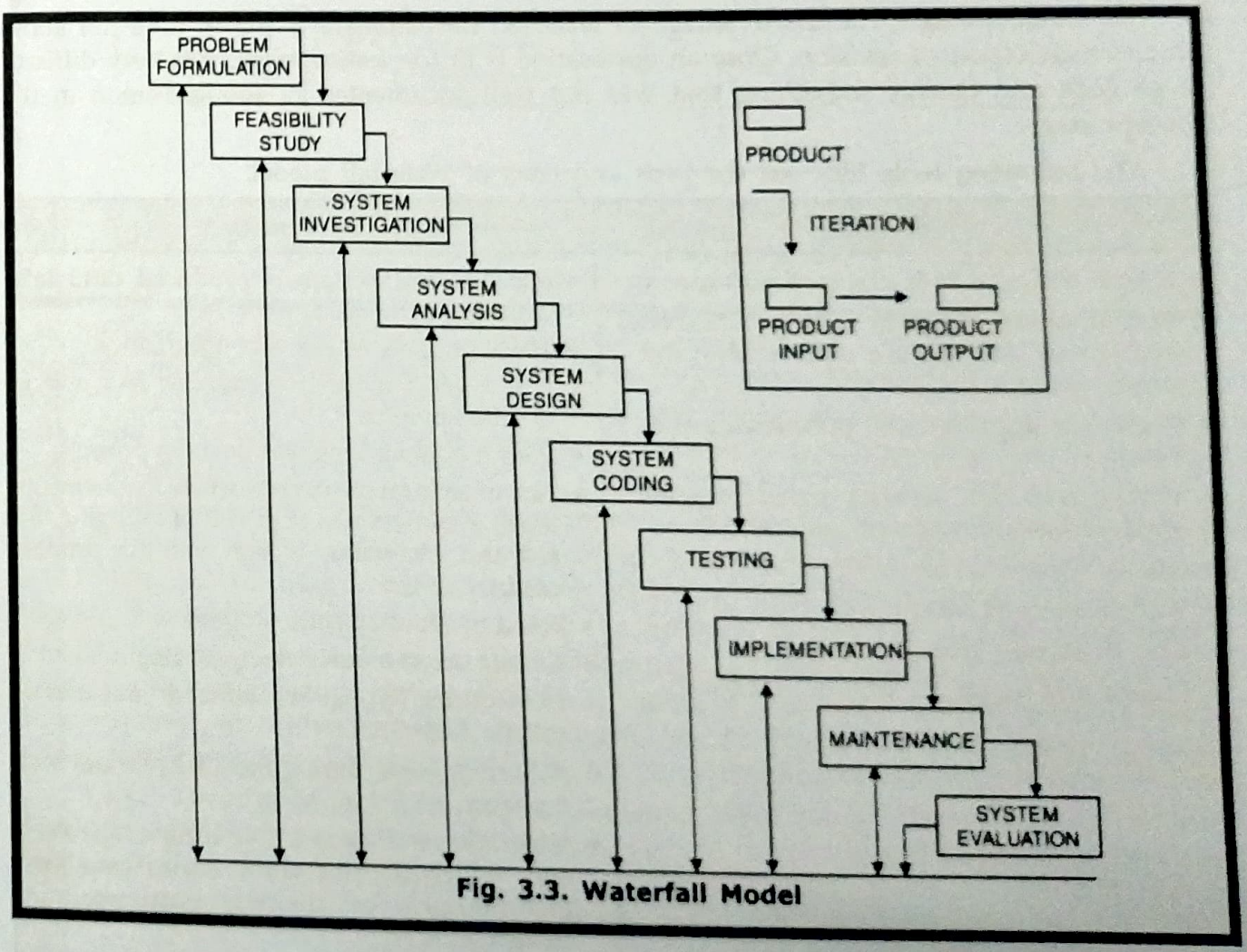

Fig. 3.3. Waterfall Model

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.

3.3.3.2. 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.

3.3.3.3. Waterfall Model Pros & Cons:

(a) Advantage: The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

(b) Disadvantage: The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The following table lists out the pros and cons of Waterfall model:

Pros	Cons
• 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 docu-mented.	• No working System 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. • No working System is produced until late in the life cycle. • 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.

3.3.4 Iterative Model

In Iterative model, iterative process starts with a simple implementation of a small set of the Software requirements and iteratively enhances the evolving versions until the complete Software is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the Software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the Software at the end of each iteration of the model.

3.3.4.1. Iterative Model design: Iterative process starts with a simple implementation of a subset of the Software requirements and iteratively enhances the evolving versions until the full Software is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a Software through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:

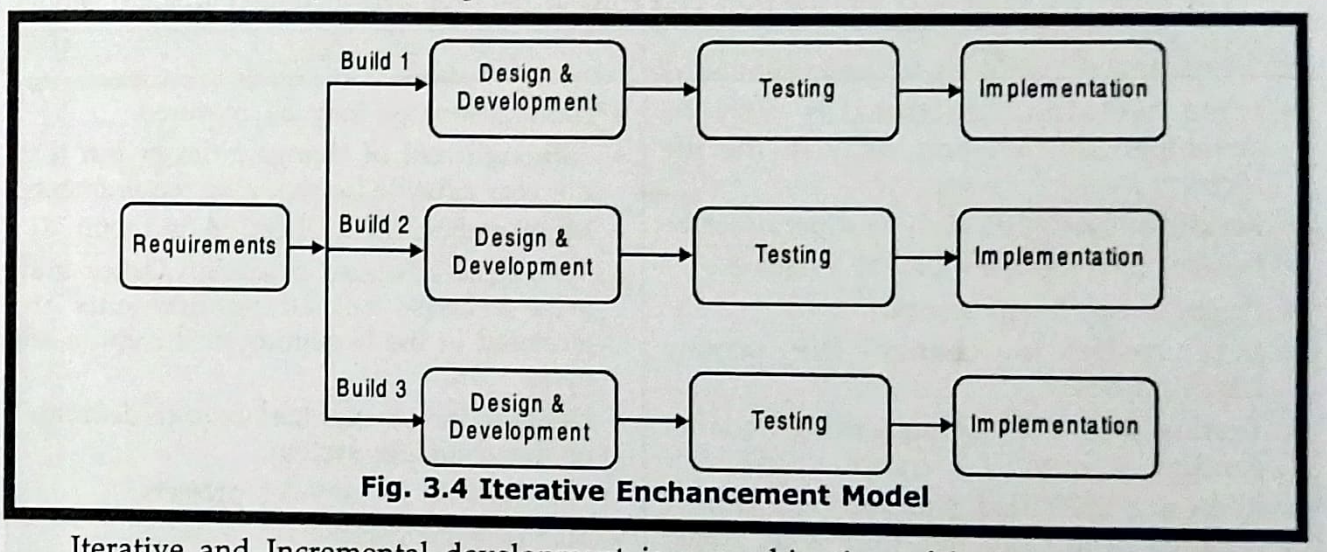

Fig. 3.4 Iterative Enchancement Model

Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During Software development, more than one iteration of the Software development cycle may be in progress at the same time." and "This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In incremental model the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete Software is ready as per the requirement.

The key to successful use of an iterative Software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the Software against those requirements within each cycle of the model. As the Software evolves through successive cycles, tests have to be repeated and extended to verify each version of the Software.

3.3.4.2. Iterative Model Application: Like other SDLC models, Iterative and incremental development has some specific applications in the Software industry. This model is most often used in the following scenarios:

- Requirements of the complete Software are clearly defined and understood.

- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

3.3.4.3. Iterative Model Pros and Cons: The advantage of this model is that there is a working model of the Software at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky Software development projects. This is because it is hard to break a small Software Software into further small serviceable increments/modules.

The following table lists out the pros and cons of Iterative and Incremental SDLC Model:

Pros	Cons
• Some working functionality can be developed quickly and early in the life cycle. • Results are obtained early and periodically. • Parallel development can be planned. • Progress can be measured. • Less costly to change the scope/ requirements. • Testing and debugging during smaller iteration is easy. • Risks are identified and resolved during iteration; and each iteration is an easily managed milestone. • Easier to manage risk - High risk part is done first. • With every increment operational product is delivered. • Issues, challenges & risks identified from each increment can be utilized/applied to the next increment. • Risk analysis is better. • It supports changing requirements. • Initial Operating time is less. • Better suited for large and mission-critical projects. • During life cycle System is produced early which facilitates customer evaluation and feedback.	• More resources may be required. • Although cost of change is lesser but it is not very suitable for changing requirements. • More management attention is required. • System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle. • Defining increments may require definition of the complete system. • Not suitable for smaller projects. • Management complexity is more. • End of project may not be known which is a risk. • Highly skilled resources are required for risk analysis. • Project.s progress is highly dependent upon the risk analysis phase.

3.3.5 Spiral Model

The spiral model combines the idea of iterative development with the Softwareatic, controlled aspects of the waterfall model. Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

3.3.5.1. Spiral Model design: The spiral model has four phases. A Software project repeatedly passes through these phases in iterations called Spirals.

- **Identification:** This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of Software requirements, subSoftware requirements and unit requirements are all done in this phase.

  This also includes understanding the Software requirements by continuous communication between the customer and the Software analyst. At the end of the spiral the product is deployed in the identified market.

Following is a diagrammatic representation of spiral model listing the activities in each phase:

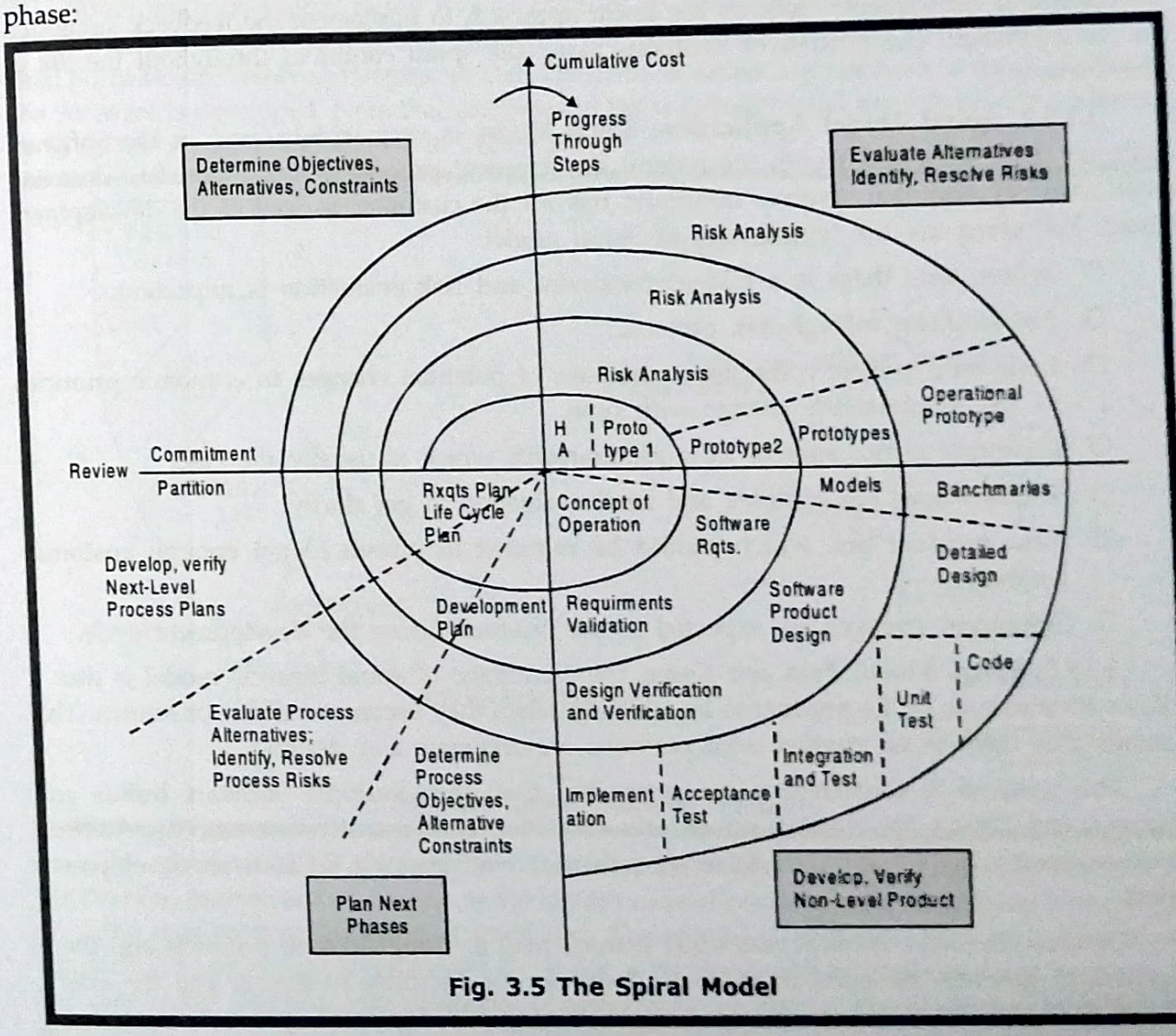

Fig. 3.5 The Spiral Model

- **Design:** Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.
- **Construct or Build:** Construct phase refers to production of the actual Software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

  Then in the subsequent spirals with higher clarity on requirements and design details a working model of the Software called build is produced with a version number. These builds are sent to customer for feedback.
- **Evaluation and Risk Analysis:** Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the Software and provides feedback.

Based on the customer evaluation, Software development process enters into the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the Software.

3.3.5.2. Spiral Model Application: Spiral Model is very widely used in the Software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the customer as well as the development firms. Following are the typical uses of Spiral model:

- When costs there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

3.3.5.3. Spiral Model Pros and Cons: The advantage of spiral lifecycle model is that it allows for elements of the product to be added in when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple Software builds and releases and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the spiral model forces early user involvement in the Software development effort.

On the other side, it takes very strict management to complete such products and there is a risk of running the spiral in indefinite loop. So the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully

The following table lists out the pros and cons of Spiral SDLC Model:

Pros	Cons
• Changing requirements can be accommodated. • Allows for extensive use of prototypes • Requirements can be captured more accurately. • Users see the system early. • Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.	• Management is more complex. • End of project may not be known early. • Not suitable for small or low risk projects and could be expensive for small projects. • Process is complex • Spiral may go indefinitely. • Large number of intermediate stages requires excessive documentation.

3.3.6. Difference between Spiral and Waterfall Models

Spiral model is similar to the waterfall model, as Software requirements are understood at early stages in both the models. However, the major risks involved with developing the final Software are resolved in spiral model. When these issues are resolved, a detail design of the Software is developed. Note that processes in the waterfall model are followed by different cycle in the spiral model shown in Figure 3.6.

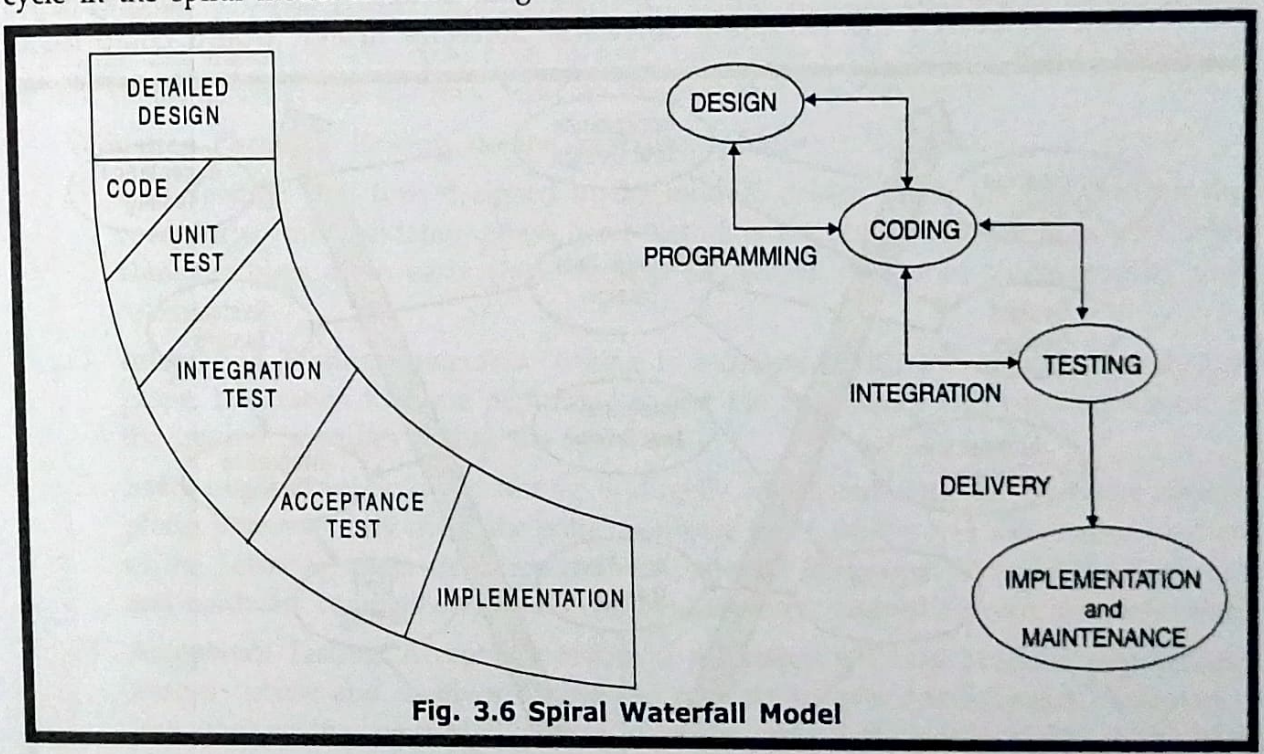

Fig. 3.6 Spiral Waterfall Model

The spiral model is also similar to prototyping model. As one of the key features of prototyping process model. As one of the key features of prototyping is to develop a prototype until the user requirements are accomplished. The second step of the spiral model functions similarly. The prototype is developed to clearly understand and achieve user requirements. If the user is not satisfied with prototype, a new prototype known as operational prototype is developed.

3.3.7 The V-Model

The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

3.3.7.1. V - Model design: Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the V and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

The below figure illustrates the different phases in V-Model of SDLC.

Verification Phases: Following are the Verification phases in V-Model:

- **Business Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
- **Software Design:** Once you have the clear and detailed product requirements, it.s time to design the complete Software. Software design would comprise of

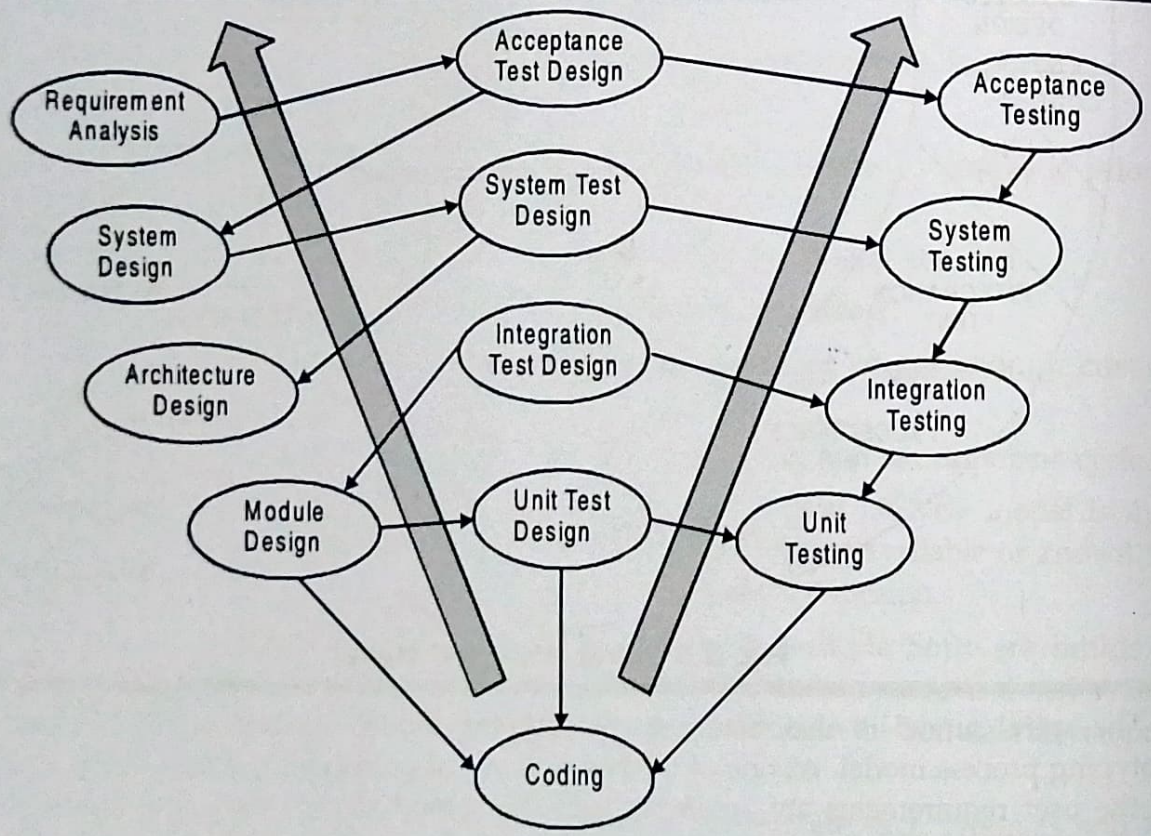

Fig. 3.7 The V-Model

understanding and detailing the complete hardware and communication setup for the product under development. Software test plan is developed based on the Software design. Doing this at an earlier stage leaves more time for actual test execution later.

- **Architectural Design:** Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. Software design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

  The data transfer and communication between the internal modules and with the outside world (other Softwares) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

- **Module Design:**In this phase the detailed internal design for all the Software modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the Software architecture and the other external Softwares. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Coding Phase: The actual coding of the Software modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the Software and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases: Following are the Validation phases in V-Model:

- **Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.
- **Integration Testing:** Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the Software.
- **Software Testing:** Software testing is directly associated with the Software design phase. Software tests check the entire Software functionality and the communication of the Software under development with external Softwares. Most of the Software and hardware compatibility issues can be uncovered during Software test execution.
- **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other Softwares available in the user environment. It also discovers the non functional issues such as load and performance defects in the actual user environment.

3.3.7.2. V - Model Application: V- Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is

used in the medical development field, as it is strictly disciplined domain. Following are the suitable scenarios to use V-Model:

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

3.3.7.3. V - Model Pros and Cons: The advantage of V-Model is that it.s very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today.s dynamic world, it becomes very expensive to make the change.

The following table lists out the pros and cons of V-Model:

Pros	Cons
• This is a highly disciplined model and Phases are completed one at a time. • Works well for smaller projects where requirements are very well understood. • 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.	• High 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. • Once an application is in the testing stage, it is difficult to go back and change a functionality • No working System is produced until late during the life cycle.

3.3.8 Big Bang Model

The Big Bang model is SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the Software developed which may or may not be as per customer requirement.

B ig Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

3.3.8.1. Big Bang Model design and Application: Big bang model comprises of focusing all the possible resources in Software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete Software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It.s an ideal model for the product where requirements are not well understood and the final release date is not given.

3.3.8.2. Big Bang Model Pros and Cons: The advantage of Big Bang is that its very simple and requires very little or no planning. Easy to mange and no formal procedure are required.

However the Big Bang model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

Following table lists out the pros and cons of Big Bang Model:

Pros	Cons
• This is a very simple model • Little or no planning required • Easy to manage • Very few resources required • Gives flexibility to developers • Is a good learning aid for new comers or students	• Very High risk and uncertainty. • Not a good model for complex and object-oriented projects. • Poor model for long and ongoing projects. • Can turn out to be very expensive if requirements are misunderstood

3.3.9 Agile Model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working Software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

At the end of the iteration a working product is displayed to the customer and important stakeholders.

3.3.9.1. What is Agile?: Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working Software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:

Agile thought process had started early in the Software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Softwares Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles

- **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working Software** - Demo working Software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
- **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** - agile development is focused on quick responses to change and continuous development.

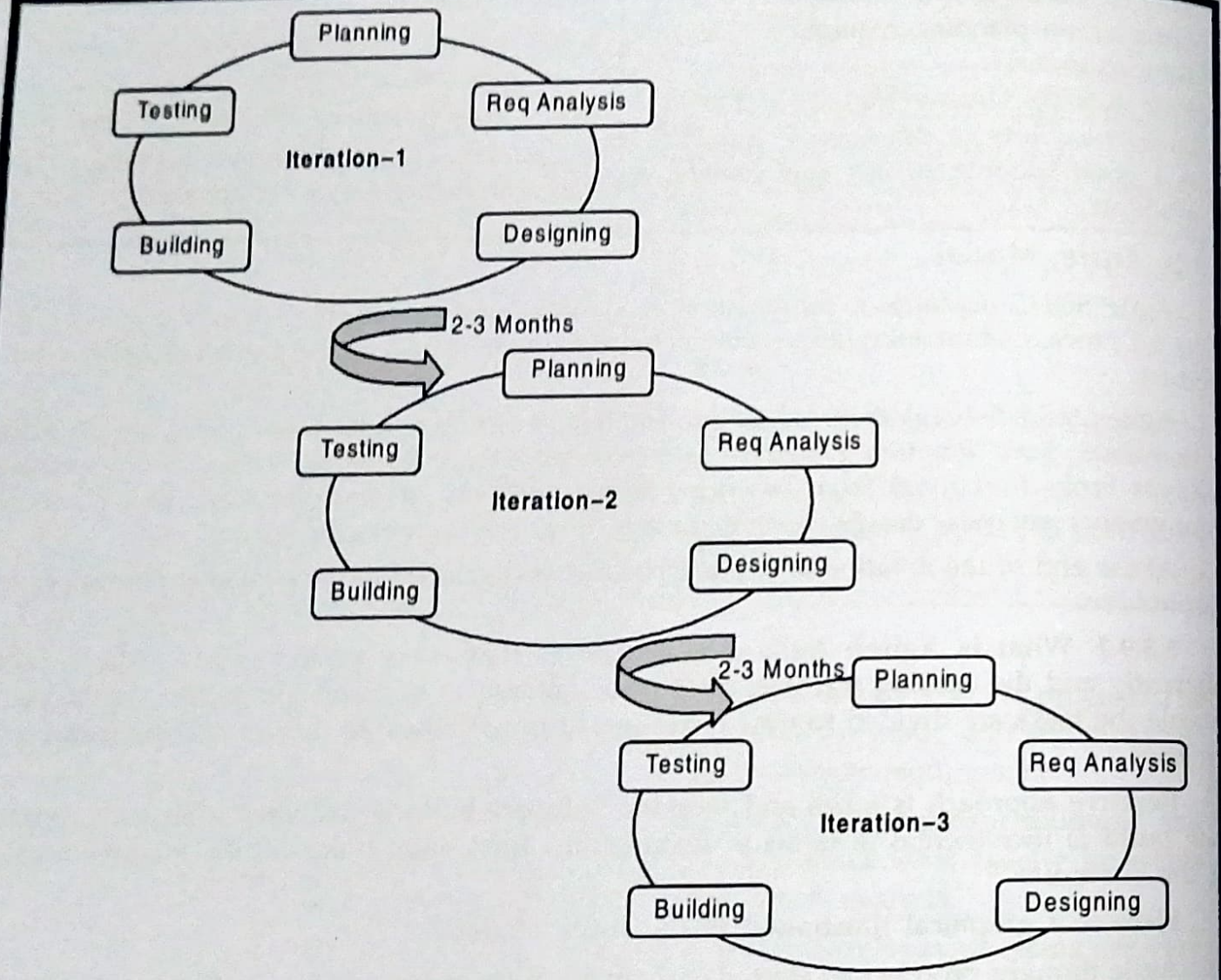

Fig. 3.8 The Agile Model

3.3.9.2. Agile Vs Traditional SDLC Models: Agile is based on the adaptive Software evelopment methods where as the traditional SDLC models like waterfall model is based on redictive approach.

Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle. Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

3.3.9.3. Agile Model Pros and Cons: Agile methods are being widely accepted in the Software world recently, however, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Following table lists out the pros and cons of Agile Model:

Pros	Cons
• Is a very realistic approach to System development • Promotes teamwork and cross training. • Functionality can be developed rapidly and demonstrated. • Resource requirements are minimum. • Suitable for fixed or changing requirements • Delivers early partial working solutions. • Good model for environments that change steadily. • Minimal rules, documentation easily employed. • Enables concurrent development and delivery within an overall planned context. • Little or no planning required • Easy to manage • Gives flexibility to developers	• Not suitable for handling complex dependencies. • More risk of sustainability, maintaina-bility and extensibility. • An overall plan, an agile leader and agile PM practice is a must without which it will not work. • Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. • Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. • There is very high individual dependency, since there is minimum documentation generated. • Transfer of technology to new team members may be quite challenging due to lack of documentation.

3.3.10 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.

3.3.10.1. What is RAD?: Rapid application development (RAD) 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 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.

3.3.10.2. RAD Model Design: RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles. Following are the phases of RAD Model:

- **Business Modeling:** 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 Modeling:** The information gathered in the Business Modeling 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 Modeling:** The data object sets defined in the Data Modeling 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 Software 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 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.

Following image illustrates the RAD Model

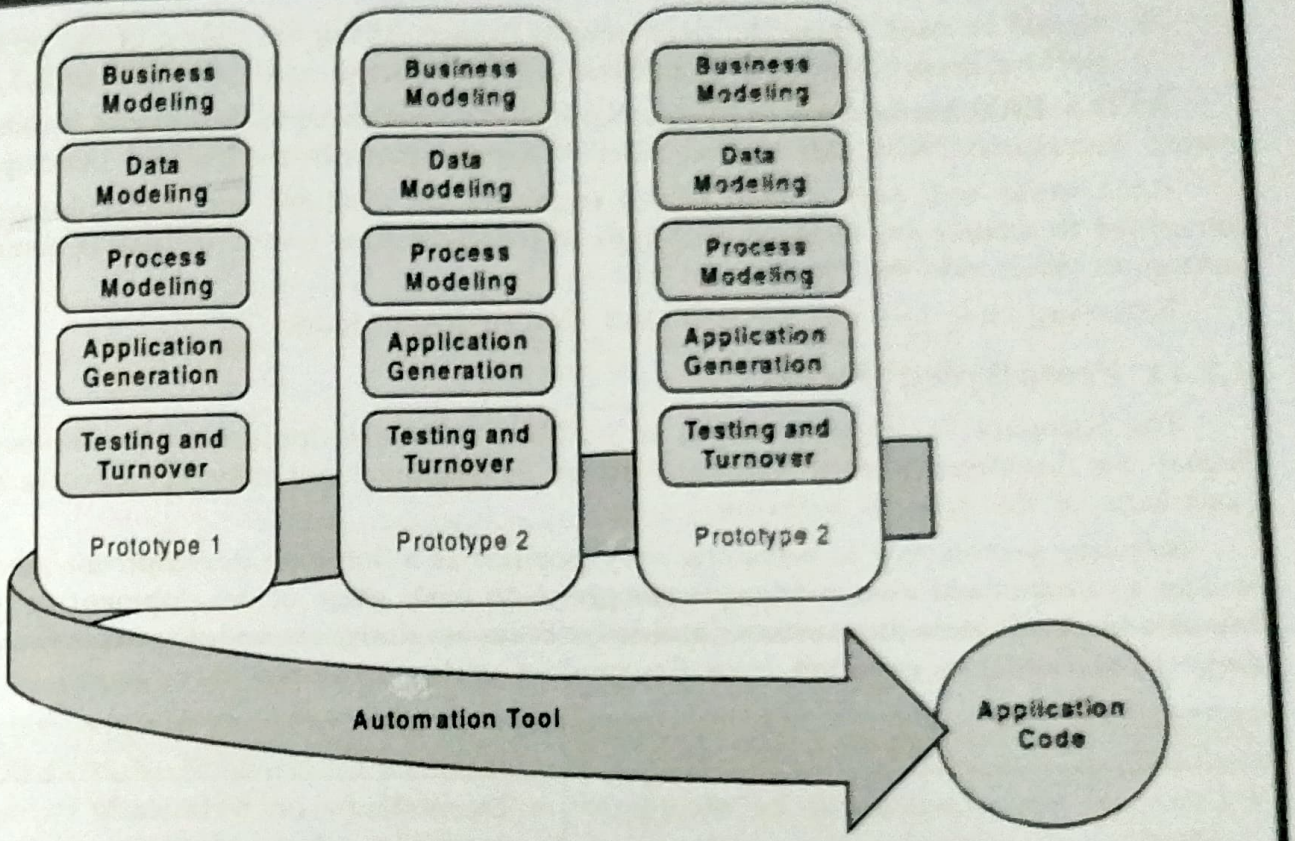

Fig. 3.9 The RAD Model

3.3.10.3. RAD Model Vs Traditional SDLC: The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn.t get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he actually gets to see the Software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.

RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

3.3.10.4. 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. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a Software can be modularized to be delivered in incremental manner.
- It should be used if there high availability of designers for modeling.
- It should be used only if the budget permits use of automated code generating tools.

concentrating on internal functions. A vertical prototype on the other side is a detailed elaboration of a specific function or a sub Software 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 Softwares. For example, database requirements, interaction and data processing loads in a given sub Software.

3.3.11.2. 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 Software 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 Software is built. Using evolutionary prototyping only 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 Softwares and then integrating all the available prototypes to form a complete Software.
- **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.

3.3.11.3. Prototyping Application: Software Prototyping is most useful in development of Softwares having high level of user interactions such as online Softwares. Softwares 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.

3.3.11.4. Prototyping Pros and Cons: Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final Software developed. The model has its own pros and cons discussed as below.

Following table lists out the pros and cons of Big Bang Model:

Pros	Cons
• Increased user involvement in the product even before 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	• Risk of insufficient requirement analysis owing to too much dependency on 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 its not technically feasible • The effort invested in building prototypes may be too much if not monitored properly

3.3.12 Build and Fix Model

In the **build and fix model** (also referred to as an **ad hoc model**), the Software is developed without any specification or design. An initial product is built, which is then repeatedly modified until it (Software) satisfies the user. That is, the Software is developed and delivered to the user. The user checks whether the desired functions 'are present. If not, then the Software is changed according to the needs by adding, modifying or deleting functions. This process goes on until the user feels that the Software can be used productively. However, the lack of design requirements and repeated modifications result in loss of acceptability of Software. Thus, Software engineers are strongly discouraged from using this development approach.

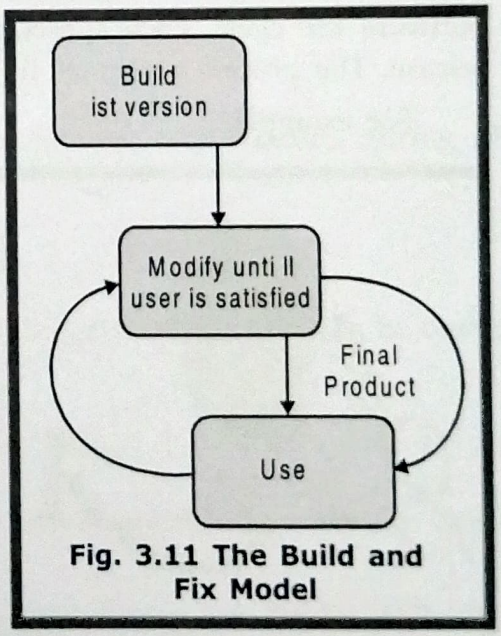

Fig. 3.11 The Build and Fix Model

This model includes the following two phases.

- **Build:** In this phase, the Software code is developed and passed on to the next phase.
- **Fix:** In this phase, the code developed in the build phase is made error free. Also, in addition to the corrections to the code, the code is modified according to the user's requirements.

Various advantages and disadvantages associated with the build and fix model are listed in Table.

Table 3.1. Advantages and Disadvantages of Build and Fix Model

Pros	Cons
• Requires less experience to execute or manage other than the ability to program. • Suitable for smaller System. • Requires less project planning.	• No real means is available of assessing the progress, quality, and risks. • Cost of using this process model is high as it requires rework until user's requirements are accomplished. • Informal design of the System as it involves unplanned procedure. • Maintenance of these models is problematic.

3.3.13 Incremental Model

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases. A working version of Software is produced during the first module, so you have working Software early on during the Software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete Software is achieved.

For example:

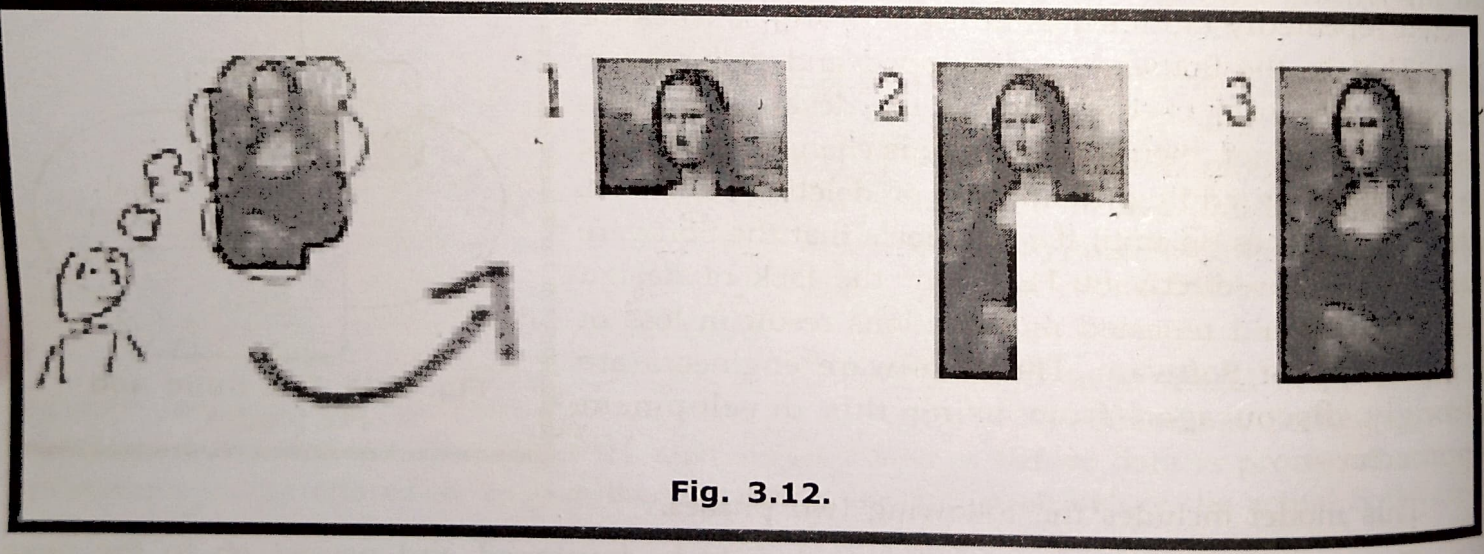

Fig. 3.12.

In the diagram above when we work **incrementally** we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it's complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third iteration the whole product is ready and integrated. Hence, the product got ready step by step.

Diagram of Incremental model:

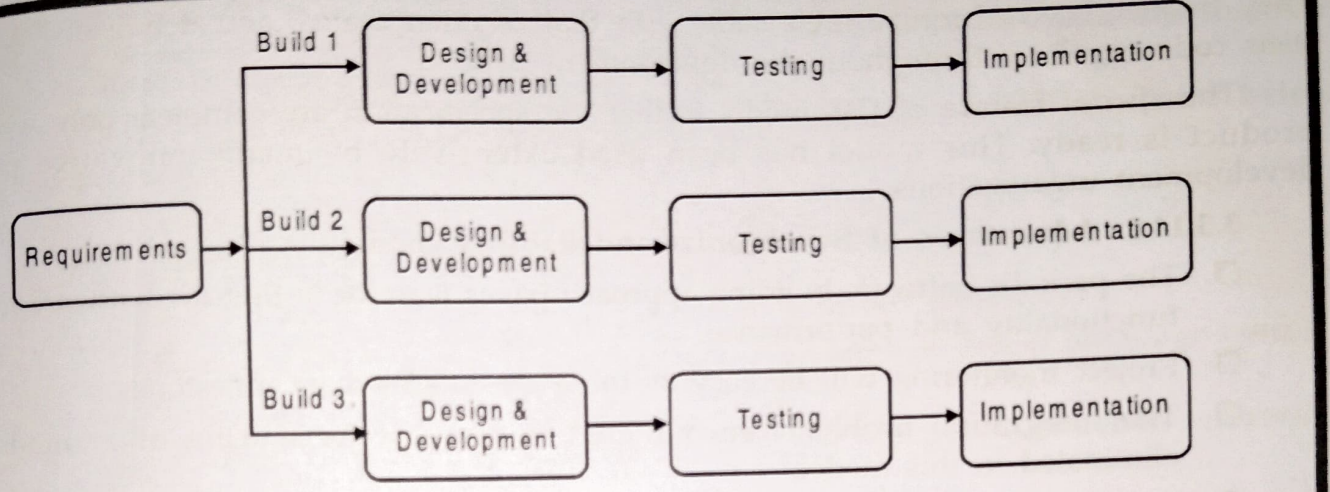

Fig. 3.13 Incremental Life Cycle Model

3.3.13.1. Advantages of Incremental model:

- Generates working Software quickly and early during the Software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

3.3.13.2. Disadvantages of Incremental model

- Needs good planning and design.
- Needs a clear and complete definition of the whole Software before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

3.3.13.3. When to use the Incremental model:

- This model can be used when the requirements of the complete Software are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

3.3.14 Synchronize-and-Stabilize Model

The synchronize and stabilize lifecycle model defines an overall approach for developing and managing large-scale Software Softwares.

Synchronize-and-stabilize (also called sync-and-stabilize) is a Softwares Development Life Cycle methodology in which teams work concurrently on individual application modules. They frequently synchronize their code with that of other teams, and debug or "stabilize" their code regularly throughout the development process.

The special feature of this model is that the specification are complete only when the product is ready. This model has been used extensively by many innovative product development organizations.

3.3.14.1. Advantages of Synchronize-and-stabilize model:

- The periodic Software building approach paves way for testing the Software for both functionality and performance.
- Project monitoring will be easy as there are intermediate milestones.
- The integration problems encountered in large projects using other models are eliminated in this model.
- Because of intermediate releases, the product can be made feature rich by incorporating the necessary feedback.

3.3.14.2. Disadvantages of Synchronize-and-stabilize model:

- A parallel independent testing team needs to be in place.
- The detailed specifications document will be made available only at time of release.
- Periodic Software builds require a rigorous process to be defined for integration of various modules.

3.3.15 Evolutionary Model

It is also called *successive versions model* or *incremental model*. At first, a simple working model is built. Subsequently it undergoes functional improvements & we keep on adding new functions till the desired Software is built.

3.3.15.1. Applications:

- Large projects where you can easily find modules for incremental implementation. Often used when the customer wants to start using the core features rather than waiting for the full Software.
- Also used in object oriented Software development because the Software can be easily portioned into units in terms of objects.

3.3.15.2. Advantages:

- User gets a chance to experiment partially developed Software
- Reduce the error because the core modules get tested thoroughly.

3.3.15.3. Disadvantages:

- It is difficult to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented & delivered.

3.2.16 Generation Techniques (FGT)

Fourth Generation techniques enable engineers to specify characteristics of software at high level and then automatically generate the source code. In addition to bring a process model, fourth generation techniques are a collection of software tools used by software

engineers to solve a problem by using a specialized language or graphic notation so that the users easily understand the problems. Hence, fourth generation techniques use instruments similar to spoken languages to allow the programmes to define what they want the computer to do rather than how to do it.

For this fourth generation techniques, certain tools are used which are listed below:

1. Non-procedural languages for database query.
2. Report generation
3. Data manipulation
4. Screen interaction and definition
5. Code generation
6. High level graphic capability
7. Spreadsheet capability
8. Automated generation of hyper text markup language and similar language used for website creation using advanced software tools.

The various advantages and disadvantages associated with fourth generation techniques are described below :

Advantages :

1. Development time is reduced, when used for small and intermediate applications.
2. The interaction between user and developer helps in detection of error.
3. When integrated with CASE tools and code generations, fourth generation techniques provide a solution to most of the software engineering problems.

Disadvantages :

1. Difficult to use
2. Restricted to small business information Software.

3.3.17 Comparison of different life-cycle models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model cannot be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used Software development model evolved so far. This model is simple to understand and use. However this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the Software is acceptable to the customer.

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging Software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

The different Software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment. On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new Software. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

POINTS TO REMEMBER

1. *Software planning is important for an organisation to achieve good results within a specified period of time.*
2. *Requirement phase includes a set of tasks which specifies the impact of the Software on the organisation.*
3. *The determination of Software requirements is a primary step for Software development life cycle which is also known as software requirement specification.*
4. *Data and Fact Gathering Techniques include research, interviews, meeting questionnaires, sampling and other techniques to collect information about the Software requirement.*
5. *The Analyst may observe the existing Software personally by following transaction.*
6. *Group interviews or meetings are held when there is not enough time to conduct personal interviews.*
7. *On-site or Direct Observation is used by the Software analyst to gain information which he cannot find by other fact finding method.*
8. *An analyst acts as a researcher, during fact finding.*
9. *Project Management and control is an essential step for the completion of a Software.*
10. *Prototype is a working model based on interaction between analysts and users.*

EXERCISE

What do you mean by Software Selection Plan Proposal ?

What is Prototyping ? Explain its advantages.

What is Software Product ?

What are Software Process Models ?

What is Classical Waterfall Model ?

What is waterfall model ?

Define the Software Process ? What is its need ?

What are the various activities of Software Process ?

9. Explain Build and fix model. Give its Advantages and Disadvantages.
10. What are the different phases of waterfall life cycle ?
11. What are the advantages and disadvantages of waterfall model ?
12. What do you know about Iterative life cycle models ? Write down phases of his model. Give its advantages and disadvantages ?
13. What is the role of Incremental Model ? Write down all the phases of this model. Give its advantages in disadvantages.
14. Explain the Spiral model with its advantages and disadvantages ?
15. How Evolutionary Model work. Also write down all the advantages and disadvantages of this model ?
16. What is the role of Prototype model in the Software development ? Give its advantages and disadvantages ?
17. How RAD Model Works ?
18. What is the difference between Sprial and Water fall.
19. Explain V Model in detail.
20. What is Big Bang Model ? Give its advantages in disadvantages.
21. How Agile Model works ?
22. What is Build and fix Mode in Software development ?