# Chapter 2

# Some Well Known Software Development Life Cycle Models

## 2.1 Introduction

It is really tough to draw a sharp line between software development approaches and Software Development Life Cycle (SDLC) models. In many literature of software engineering, these terms are even used interchangeably. So, before we begin the details discussion of the topic, let us somehow draw the boundary line between software development approaches and SDLC process models. Defining these two terms are beyond the scope of this chapter, however we just try to explain the both only to establish the differences from our point of view.

### 2.1.1 Software Development Life Cycle or Software Process

The terms Software Development Life Cycle (SDLC) and software process are used interchangeably or in conjunction. A SDLC spans over the time period from the concept development to the product retirement. Software development life cycle involves the step-by-step activities in the development of a software product. The whole process is generally classified into a set of steps and a specific operation will be carried out in each of the steps. SW development process or simply process typically defines the set steps to be carried out during the development of the system.

### 2.1.2 Software Development Life Cycle Models or Process Models

The SDLC process model typically depicts the fashions in which the SW process to be carried out i.e. which steps to be done before or after another step. A software develop-

ment process model is an approach to the SDLC that describes the sequence of steps to be followed while developing software projects (7, 8). In general all the process models do cover all distinct phases defined by SW process, but in different manner or sequence– which makes one process model differ from the other.

**a. Purposes for articulating software life cycle models**

There are a variety of purposes for articulating software life cycle (SDLC) models. SDLC model serve as a (9, 10, 11) :

- guideline to organize, plan, staff, budget, schedule and manage software project work over organizational time, space and computing environments
- prescriptive outline for what documents to produce for delivery to client
- basis for determining what software engineering tools and methodologies will be most appropriate to support different life cycle activities
- framework for analyzing or estimating patterns of resource allocation and consumption during the software life cycle
- basis for conducting empirical studies to determine what affects software productivity, cost, and overall quality

**b. Types of SDLC Process Models**

A software life cycle model is basically the characterization of how software is or should be developed. The SDLC process model can be either Perspective of Descriptive.

1. ***Prescriptive* SDLC Process Model :** A *prescriptive model* prescribes how a new software system should be developed and are used as guidelines or frameworks to organize and structure how software development activities should be performed in specific order. Typically, it is easier and more common to articulate a prescriptive life cycle model for how software systems should be developed. This is possible since most such models are intuitive or well reasoned. This means that many idiosyncratic details that describe how a software system is built in practice can be ignored, generalized or deferred for later consideration. This, of course, should raise concern for the relative validity and robustness of such life cycle models when developing different kinds of application systems in different kinds of development settings, using different programming languages with differentially skilled staff etc. However, prescriptive models are also used to package the development tasks and

techniques for using a given set of software engineering tools or environment during a development project.

2. ***Descriptive* SDLC Process Model :** A *descriptive model* describes the history and characterize how particular software systems are actually developed in specific settings. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models (12). As such, they are less common and more difficult to articulate for obvious reasons:
   - one must observe or collect data throughout the life cycle of a software system, a period of elapsed time often measured in years.
   - also, descriptive models are specific to the systems observed and only can be generalize through systematic comparative analysis.

Therefore, this suggests the prescriptive software life cycle models will dominate attention until a sufficient base of observational data is available to articulate empirically grounded descriptive life cycle models.

### 2.1.3 Software Development Approaches or Philosophies

Several software development philosophies have been proposed by different authors over the time to address the challenges in software development. The most common software development philosophies are like iterative, incremental, agile, evolutionary and component based development philosophies. These software development philosophies can be implemented following other process models i.e. Waterfall, RAD, Spiral, Prototyping or alike for better and optimal results. These software development philosophies are discussed briefly in the subsequent sections.

## 2.2 Origin of Software Development Lifecycle Models

In early days, software engineering approach was *ad hoc*. Around 1970s, introduction of *structured programming* gave a formal shift in software engineering from the ad hoc to a systematic approach. Then around 1980s, introduction of *object oriented programming* with some advancement explores new areas in software engineering. In recent dates, with the introduction of *Component Based Software Development (CBSD)*, the industry

is moving in a new direction. These day's software systems are more complex as compared to those of early. These complex, high quality software systems are built efficiently using component based approach in a shorter time. The importance of component based development lies in its efficiency. In the remaining part of this chapter the term *component* and *software components* will be used interchangeably.
Explicit models of software evolution date back to the earliest projects developing large software systems in the 1950s and 1960s (13, 14). Overall, the apparent purpose of these early software life cycle models was to provide a conceptual scheme for rationally managing the development of software systems. Such a scheme could therefore serve as a basis for planning, organizing, staffing, coordinating, budgeting, and directing software development activities.

## 2.3 Typical Phases of Software Development Life Cycle

The typical steps of any Software (SW) Development Life Cycle Models are as follows :

1. Project Planning
2. Risk Analysis
3. Software Requirement Gathering, Analysis, and Specification
4. Feasibility Study
5. System Analysis and Design
6. Implementation or Code Generation
7. Testing
8. System Deployment and Operation
9. Maintenance and Support
10. Customer Evaluation

Each of the steps has its own importance and plays a significant role during the product development. As description of each of the steps can give a better understanding, we briefly discussed these phases below :

1. **Project Planning :** This is the first and foremost important stage in the development. The basic motive is to plan the total project and to estimate the merits and demerits of the project. The planning phase includes the definition of the intended system, development of the project plan and parallel management of the plan throughout the proceedings. A good and matured plan can create a very good initiative and can positively affect the complete project. In this phase, the objectives, alternatives and constraints of the project are determined and documented. The objectives and other specifications are fixed in order to decide which strategies or approaches to follow during the project life cycle.

2. **Risk Analysis :** In this phase, all possible alternatives which can help in developing a cost effective project are analyzed and strategies are decided so as to use them. This phase has been added specially in order to identify and resolve all the possible risks in the project development. If risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out a possible solution in order to deal with the potential changes in the requirements.

3. **Software Requirement Gathering, Analysis, and Specification :** The essential purpose of this phase is to find the need and to define the problem that needs to be solved. As software is always of a large system, before software companies start working on any project, they do an in-depth analysis of client requirements for the project. This helps to suggest the best solution which will suit customers'current requirements and also which is capable of scaling to accommodate the future requirements as well. System is the basic and very critical requirement for the existence of software in any entity. So if the system is not in place, the system should be engineered and put in place. In some cases, to extract the maximum output, the system should be re-engineered and spruced up. The project work begins by establishing the requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when the software must interface with other elements such as hardware, people and other resources. The *requirement gathering* process is intensified and focused specially on software. To understand the nature of the program(s) to be built, the system engineer or system analyst must understand the information domain for the software, as well as required functions, behavior, performance and interfacing. In this phase, the development team visits the customer and studies their system. They investigate

the need for possible software automation in the given system. The main aim of the *analysis phase* is to perform statistics and requirements gathering. Based on the analysis of the project and due to the influence of the results of the planning phase, the requirements for the project are decided and gathered. Once the ideal system is engineered or tuned, the development team studies the software requirement for the system.

In the *requirement specification* activity, after the requirements for the project are analyzed, they are prioritized and made ready by specifying in a document called *System Requirement Specification* (SRS) for further use.

4. **Feasibility Study:** In this phase, the various feasibilities i.e. economic, operational, technical and environmental feasibilities of the project are analyzed. By the end of the feasibility study, the team furnishes a document that holds the different specific recommendations for the candidate system. It also includes the personnel assignments, costs, project schedule, target dates etc. Most of the developers have the habit of developing a prototype of the entire software and represent the software as a miniature model. The flaws, both technical and design, can be found and removed and the entire process can be redesigned.

5. **System Analysis and Design :** Once the analysis is over, the design phase begins. This is the first phase when a project kicks-off. The aim is to create the architecture of the total system following a modular approach. In this phase, the overall structure and its nuances are defined for the software. This is one of the important stages of the process and serves to be a benchmark stage since the errors performed until this stage and during this stage can be rectified here. Any glitch in the design phase could be very expensive to solve in the later stage of the software development. During this phase, software companies come up with the designs for the front-end as well as a scalable and robust application framework including the database design. In terms of the client/server technology, the *number of tiers needed* for the package *architecture*, the *database design*, the *data structure design* etc... are all defined in this phase. Thus a software development model is created and the logical system of the product is developed in this phase.

6. **Implementation/Code Generation :** Once the design is finalized, reviewed and approved then the design must be translated into a machine readable form using the appropriate programming languages like C, C++, Pascal, Java, J2EE, etc. as per the project type and its requirements. The application development actually

starts now.

The implementation or code generation step performs this task. If the design is performed in a detailed manner, code generation can be accomplished without much complication. Programming tools like editors, compilers, interpreters, debuggers etc. are used to generate the code.

7. **Testing :** Once the code is generated, the software testing begins. The testing phase is one of the final stages of the development process and this is the phase where the final adjustments are made before presenting the completely developed software to the end user. In general, the testers encounter the problem of removing the logical errors and bugs. Unit testing, integration testing and system testing are done at different milestones of the project to ensure a bug free delivery. The test conditions which are decided in the analysis phase are applied to the system and if the output obtained is equal to the intended output, it means that the software is ready to be provided to the user. Different testing tools and methodologies are already available to unravel the bugs that were committed during the previous phases. Some companies build their own testing tools that are tailor made for their own development operations. Adherence to various programming and testing standards is strictly followed to ensure the code quality and performance.

8. **System Deployment and Operation :** After the completion of coding and testing, the application is deployed in the actual operational environment and it goes live. Software companies also provide different kind of application support to keep your application always on the go.

9. **Maintenance and Support :** The software will definitely undergo change once it is delivered to the customer due to many reasons. The software should be developed to accommodate changes that could happen during the post implementation period. Change could happen because of some unexpected input values into the system. Any updates required to the application, once it goes live, can be done as part of project maintenance work. Further, the changes in the system could directly affect the software operations. Thus, the toughest job is encountered in the maintenance phase which normally accounts for the highest amount of money. The maintenance team is decided such that they monitor on the change in organization of the software and report to the developers, in case a need arises. The information desk is also provided if required in this phase to maintain the relationship between the user and the creator.

10. **Customer Evaluation :** In this phase, developed product is passed on to the customer in order to receive comments and suggestions which can help in identifying and resolving potential problems/errors in the software developed. This feedback report may be used by the organizations for process improvement and knowledge enhancements through learning from their past mistakes.

Any process model may further combine one or more of these phases or even may split a single phase in to multiple sub-phases or individual phases depending on the project need and by means of this different process models differs among themselves.

## 2.4 Introduction to Software Development Approaches or Philosophies

Several software development philosophies have been proposed by different authors over the time to address the challenges in software development. The most common software development philosophies are like Iterative development, incremental development, agile development, component based development philosophies etc. These software development philosophies may be followed together with some Software Development Lifecycle Models (SDLC) for better and optimum results. In this section we shall outline these philosophies briefly.

### 2.4.1 Iterative Software Development Philosophy

An iterative Software Development does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at each iteration. The iterative software development philosophy can be likened to produce software by successive approximation. The key to successful use of an iterative software development lifecycle is rigorous validation of requirements and verification of each version of the software against those requirements within each iteration of the model. The following Figure 2.1 depicts the iterative software development philosophy :

**Features of Iterative Software Development Philosophy :**

The features of iterative approach those made it generally superior to a linear approach are:

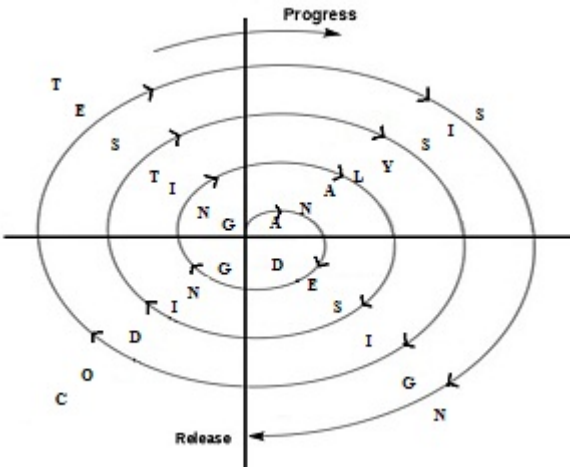


**Figure 2.1:** Iterative Software Development Philosophy

- **Requirements Change Consideration :** The truth is that requirements usually change over the lise span of a system. Iterative development lets one to take into account changing requirements. Changing requirements and requirements "creep"have always been primary sources of project trouble, which lead to late delivery, missed schedules, unsatisfied customers and frustrated developers.

- **Progressive Integration :** In iterative development, integration is not one "big bang"at the end; instead, elements are integrated progressively. The iterative approach is almost a process of continuous integration.

- **Early Risk Mitigation :** The iterative approach lets mitigate risks earlier because integration is generally the only time that risks are discovered or addressed. As you unroll the early iterations, you go through all process components, exercising many aspects of the project, such as tools, off-the-shelf software, people skills, and so on. Perceived risks will prove not to be risks, new, unsuspected risks will be revealed.

- **Making Tactical Changes :** It provides management with a means of making tactical changes to the product for whatever reason, for example, to compete with existing products. You can decide to release a product early with reduced functionality to counter a move by a competitor or one can adopt another vendor for a given technology.

- **Reuse :** It facilitates reuse because it is easy to identify common parts as they are partially designed or implemented instead of identifying all commonality in the

beginning before anything has been designed or implemented which is typically difficult. Design reviews in early iterations allow architects to identify unsuspected potential reuse and then develop, and mature common code for it in subsequent iterations.

- **Robust :** It results in a very robust architecture because you correct errors over several iterations. Flaws are detected even in the early iterations as the product moves beyond inception into elaboration rather than in one massive testing phase at the end. Performance bottlenecks are discovered at a time when they can still be addressed instead of creating panic on the eve of delivery.

- **Progressive Learning :** Developers can learn along the way, and their various abilities and specialties are employed more fully during the entire lifecycle. Testers start testing early, technical writers write early, and so on. In a non-iterative development, the same people would be waiting around to begin their work, making plan after plan but not making concrete progress. What can a tester test when the product consists only of three feet of design documentation on a shelf? Training needs or the need for additional people is spotted early during assessment reviews.

- **Improved Process :** The development process itself can be improved and refined along the way. The assessment at the end of each iteration not only looks at the status of the project from a product or schedule perspective but also analyzes what should be changed in the organization and in the process to make it perform better in the next iteration.

### 2.4.2 Incremental Software Development Philosophy

The incremental software development philosophy (as depicted in Figure 2.2) combines elements of the linear sequential model in a staggered fashion as calendar time progresses with the iterative philosophy of prototyping repeatedly. Each linear sequence produces a deliverable "increment"of the software. For example, the ATM system of any bank developed using the incremental paradigm might deliver money withdrawal and balance inquiry capabilities in the first increment; money transfer capability in the second increment; and other capabilities in the subsequent increments.

When an incremental model is used, the first increment is often a core product. The basic requirements are addressed, but many known and unknown supplementary features

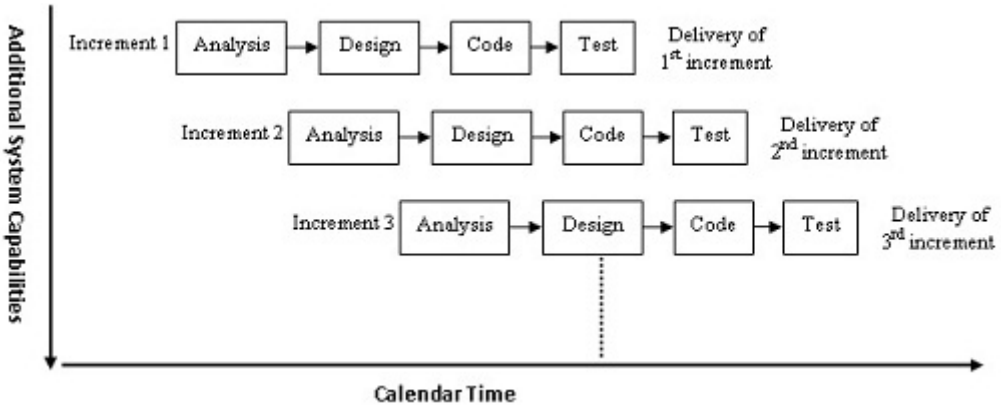


**Figure 2.2:** Incremental Software Development Philosophy

remain undelivered. The core product is used by the customer.

**Features of Incremental Software Development Philosophy :**
The incremental philosophy has the following features:

- Early increments can be developed with few people
- It combines iterative nature of prototyping model and linear nature of Linear Sequential Model
- Less number of people are required
- Improves product quality
- The system can be designed in such a manner that it can be delivered into pieces
- Increments are developed one after the other, after feedback has been received from the user
- Since each increment is simpler than the original system, it is easier to predict resources needed to accomplish the development task within acceptable accuracy bounds
- Increments can be planned to manage technical risks
- It can be applied to those projects which have independent modules

### 2.4.3 Agile Software Development Philosophy

Agile software development is a concept, a philosophy and a methodology which evolved during 1990s as an answer to the long growing frustrations of the waterfall SDLC concepts. The term promotes an iterative approach to software development using shorter and lightweight development cycles and some different deliverables. Agile software development is a philosophy for managing software projects and teams. Agile represents a group of software engineering methodologies which promise to deliver increased productivity, quality and project success rate overall in software development projects.

**Features of Agile Software Development Philosophy :**

The features of Agile software development philosophy are as follows :

- **Modularity :** Modularity allows a process to be broken into components called activities capable of transforming the vision of the software system into reality.
- **Iterative :** Agile software processes focus on short iterations or cycles typically started and completed in a matter of weeks. Within each cycle, a certain set of activities is completed. However, a single cycle will probably not be enough to complete the entire project. Therefore, the short cycle is repeated many times to refine the deliverables.
- **Time-Bound :** Time-boxing having a fixed time limit is a popular technique for bounding an iteration (15). This technique forces trade-offs to be made in the elements of the project and also encourages optimization of the agile process. One thing that time boxes are not allowed to be used for is to pressure project members into making poor decisions, working overtime, and cutting corners on quality (16).
- **Parsimony :** We must implement and test until the system works as it is intended to. These activities are not negotiable. Until someone radically changes the way that we create software, there will be a certain set of mandatory activities necessary to deliver systems. Agile software processes focus on parsimony. That is, they require a minimal number of activities necessary to mitigate risks and achieve their goals. By minimizing the number of activities, they allow developers to deliver systems against an aggressive schedule maintaining some semblance of a normal life.
- **Adaptive :** During an iteration, new risks may be exposed which require some activities that were not planned. The agile process adapts the process to attack

these new found risks. If the goal cannot be achieved using the activities planned during the iteration, new activities can be added to allow the goal to be reached. Similarly, activities may be discarded if the risks turn out to be ungrounded.

- **Incremental :** An agile process does not try to build the entire system at once. Instead, it partitions the nontrivial system into increments which may be developed in parallel at different times and at different rates. We unit test each increment independently. When an increment is completed and tested, it is integrated into the system. Each increment may require several iterations to complete.

- **Convergent :** Convergence states that we are actively attacking all of the risks worth attacking. As a result, the system becomes closer to the reality that we seek with each iteration. As risks are being proactively attacked, the system is being delivered in increments. We are doing everything within our power to ensure success in the most rapid fashion. Convergence is taken for granted in most software development processes and subsequently is often assumed. However, if risks such as "excessive feature creep"are not addressed early in the project, the project can spiral out of control. Excessive feature creep is not the same as changing requirements. Requirements will change over the course of a project. These changes are handled by the iterative and adaptive characteristics of the agile process. If requirements do change, there should be an activity in the process to capture these changes. However, a user can change his mind more rapidly than software can developed. If the requirements are not converging things are probably out of control.

- **People Oriented :** Small teams are a central theme of agile processes. Agile process empower developers by creating small teams around an increment (17). Even large projects using agile software processes divide their members into smaller teams. This gives people the sense that they are not isolated elements in a large organization (17). Increments provide the ideal deliverable for these teams.

- **Collaborative :** Communication is a vital part of any software development project. When a project is developed in pieces, understanding how the pieces fit together is vital to creating the finished product. There is more to integration than simple communication. Quickly integrating a large project while increments are being developed in parallel, requires collaboration. Agile processes foster communication among team members.

- **Complimentary :** Complimenting is an aspect of good process design that utilizes downstream activities to validate and enhance the outputs of earlier activities. Complimentary activities are activities that work together to produce a better result than they would individually. For example, the activities, "write user stories", "create acceptance tests"and "estimate the user story"are complimentary because they produce descriptions of functionality that can be tested and written within the scope on an iteration. These three activities are part of Extreme Programming.

## 2.5 Different Well Known SDLC Process Models

Many people have proposed different software development process models. Many of them are quite same in different aspects while other differs. Here we just consider some well known SDLC process models enlisted below :

- Classical Waterfall Model
- Iterative Waterfall Model
- Prototype Model
- Spiral Model
- V-Model
- RAD Model
- RUP Model
- Evolutionary Model

The details discussion of these SDLC model is beyond the scope of this thesis, but we just highlight the features of these models which are important for our considerations in the coming sections. The readers may follow the references for further detail discussion of these process models (18, 19, 20).

### 2.5.1 Classical Waterfall Model

**History of Classical Waterfall Model**

The first formal description of the classical waterfall model (shown in Figure 2.3) or simply waterfall model is often cited as a 1970 article by Winston W. Royce(21), though he did not use the term "waterfall"in his article. As an initial concept Royce presented this model as an example of a flawed, nonworking model. The phrase "waterfall model"quickly came to refer not to Royce's final, iterative design, but rather to his purely sequentially ordered model. Royce (21) started his 1970 article "Managing the development of large software systems"with a statement about the origin of his ideas : *I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.* The waterfall development model originates in the manufacturing and construction industries : highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware oriented model was simply adapted for software development. Waterfall is a software development model with strictly one Iteration/phase. In this process model, development proceeds sequentially through the phases : requirements analysis, design, coding, testing, integration, and maintenance (14).

**Features of Classical Waterfall Model**

Waterfall approach was first Process Model to be introduced and followed widely in Software Engineering to ensure success of the project. In this approach, the whole process of software development is divided into separate process phases. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirement and Analysis, Design, Code, Testing, Implementation and Maintenance(14) in a strict, planned sequence.

**Advantages of Classical Waterfall Model**

The advantages of waterfall model are :

- employs a systematic, orthodox method of project development and delivery
- simple to understand and use

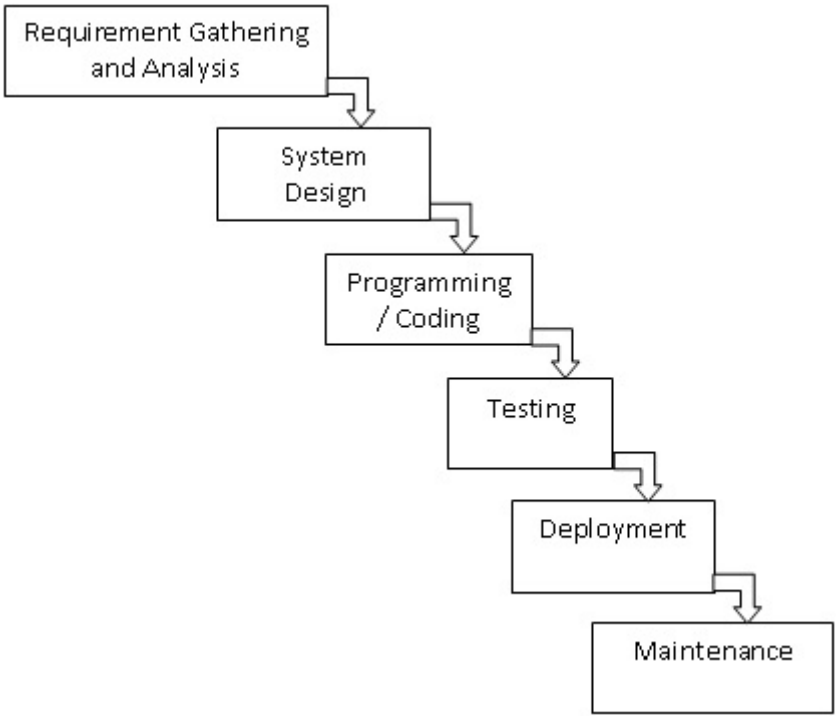


**Figure 2.3:** Classical Waterfall Model

- clearly defined stages
- clear project objectives
- strict sign-off requirements -stable project requirements
- progress of system is measurable through well understood milestones
- this model is extremely easy to understand and therefore, is implemented at various project management levels and in a number of fields
- easy to arrange tasks
- process and results are well documented
- each phase has specific deliverable and a review
- the waterfall methodology is easy on the manager
- customers/End users already know about it
- it allows for departmentalization and managerial control. Each phase of development proceeds in strict order, without any overlapping or iterative steps

- provides a disciplined and structured approach which makes it easy to keep projects under control
- limits the amount of cross team interaction needed during development

**Disadvantages of Classical Waterfall Model**

The disadvantages of waterfall model are :

- it does not allow for much reflection or revision i.e. difficulty in responding to changes
- time consuming
- customers have unreasonable time lines and expectations
- communication gaps exist between customers, engineers and project managers
- this methodology can be grueling for developers as oversights and flawed design work can seriously affect the budgeted costs and final launch date
- debugging can be complicated
- leaves no room for feedback anywhere in the process except at the end
- the methodology can give the false expectation of predictability and the reality of cost and date overruns when applied to the wrong situations
- it doesn't support iterative methodology
- being a strictly sequential model, jumping back and forth between two or more phases is impossible
- bugs and errors in the code cannot be discovered until and unless the testing phase is reached leading to wastage of time and other precious resources
- customers don't (really) know at the beginning what they want but any scope for adjustment during the life cycle can end a project
- requirements change during the course of the project is difficult
- no working software is produced until late in the life cycle– the deliverable is a onetime at the end of the complete development

- risk and uncertainty is high with this process model
- users can only judge quality at the end

**Suitability of Classical Waterfall Model**

- works well for projects where requirements are well understood but not suitable when the project requirements are dynamic or constantly changing
- works well when quality is more important than cost and schedule
- not suitable for complex projects

### 2.5.2 Iterative Waterfall Model

The primary limitation of the classical waterfall model is that it doesn't allow going back to its previous phases from any other phase. Hence, if any error or issue is detected at any phase which requires making necessary changes to its previous or earlier phase deliverables– is simply impossible with classical water fall model. In Iterative Waterfall Model (Figure 2.4) this particular limitations is overcome by adding backward path to its previous phases from any particular phase.

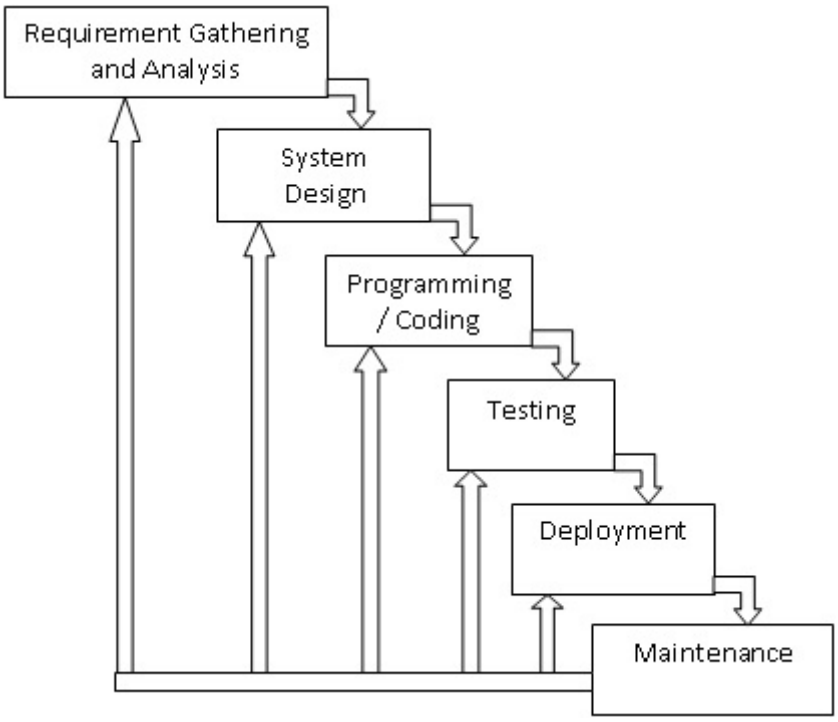


**Figure 2.4:** Iterative Waterfall Model

**Advantages of Iterative Waterfall Model**

In addition to the advantages of Classical Waterfall Model, Iterative Waterfall Model provides the following advantages :

- allows for much reflection or revision– although an application is in the testing stage, it is possible to go back and change something that was not well thought out in the concept stage facilitating responding to changes
- allows iteration and flexibility
- carries less risk than a classical Waterfall Model
- one can find and correct defects over several iterations producing a robust architecture and high quality application
- provides many opportunities during the development to learn from earlier mistakes and improve developer skills from one iteration to another ensuring team members to learn along the way
- produces better quality products

**Disadvantages of Iterative Waterfall Model**

The disadvantages of iterative waterfall model are :

- time consuming
- customers have unreasonable timelines and expectations
- communication gaps exist between customers, engineers and project managers
- when applied to the wrong situations the methodology can give the false expectation of predictability and the reality of cost and date overruns.
- users can only judge quality at the end
- very risky, since one process cannot start before finishing the other

**Suitability of Iterative Waterfall Model**

The iterative waterfall model is suitable for the types of projects when the project understanding and system requirements are not very clear at the beginning of the project.

### 2.5.3 Prototype Model

**Software Prototyping**

During 1960s and 1970s monolithic development cycle of building the entire program first and then working out any inconsistencies between design and implementation were followed. That led to higher software costs and poor estimates of time and cost. The monolithic approach has been dubbed the "Slaying the (software) Dragon"technique, since it assumes that the software designer and developer is a single hero who has to slay the entire dragon alone.

A prototype is a working model that is functionally equivalent to a component of the product which is not based on strict planning, but is an early approximation of the final product or software system and acts as a sample to test the process. From this sample we learn and try to build a better final product. A prototype typically implements only a small subset of the features of the eventual program, and the implementation may be completely different from that of the eventual product. Software prototyping activity is the creation of prototypes, i.e., incomplete versions of the software program being developed which may or may not be completely different from the final system we are trying to develop.

The purpose of a prototype is to allow users of the software to evaluate proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping can avoid the great expense and difficulty of changing a finished software product. 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. The developmental process only continues once the client is satisfied with the functioning of the prototype. At that stage the developer determines the specifications of the client's real needs.

**Features of Prototype Model**

It is a software development process that begins with requirements collection, followed by prototyping and user evaluation that facilitates to discover new or hidden requirements during the development (22). This is a cyclic version of the linear model. The goal of prototyping based development is to counter the first two limitations of the waterfall. The basic idea here 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. Development of the prototype obviously undergoes design, coding and testing. But each of these phases is not done very formally or thoroughly. 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.
Prototyping 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 such situations letting the client "plan"with the prototype provides invaluable and intangible inputs which helps in determining the requirements for the system. It is also an effective method to demonstrate the feasibility of a certain approach. This might be needed for novel systems where it is not clear those constraints can be met or that algorithms can be developed to implement the requirements. The process model of the prototyping approach is shown in the Figure 2.5.

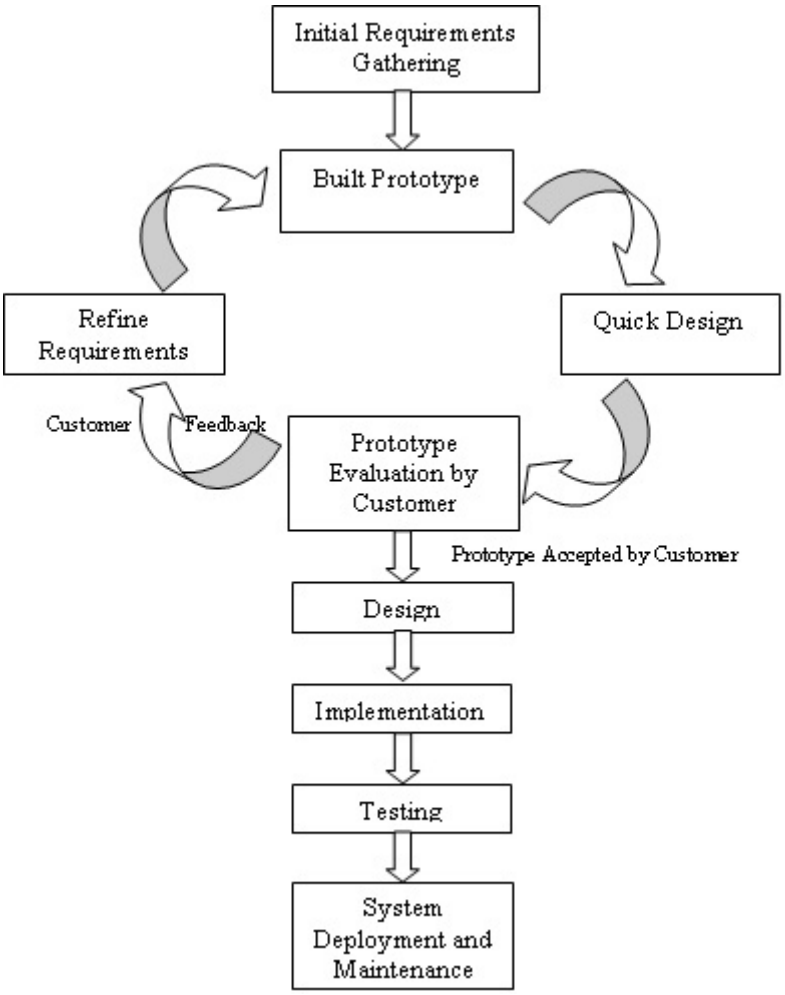


**Figure 2.5:** Prototype Model

The basic reason for little common use of prototyping is the cost involved in this

built-it-twice approach. However, some argue that prototyping need not be very costly and can actually reduce the overall development cost. The goal is to provide a system with overall functionality. On the other hand, the experience of developing the prototype will very useful for developers when developing the final system. This experience helps to reduce the cost of development of the final system and results in a more reliable and better designed system. Most of the successful software products have been developed using this model — as it is very difficult to comprehend all the requirements of a customer in one shot. There are many variations of this model tailored with respect to the project management styles of the companies. New versions of a software product evolve as a result of prototyping.

**Advantages of Prototype Model**

There are many tangable and abstract advantages of using prototyping in software development:

- **Proper clarity and 'feel'of the Proposed System :** When prototype is shown to the user, the user gets a proper clarity and 'feel'of the functionality of the software and he can suggest changes and modifications.
- **Better System Demonstration and Understanding :** Since in this methodology a working model of the system is provided, from demonstration of the prototype the users get a better understanding of the system being developed.
- **Reduces Project Risk :** It reduces project failure risks by identifying the potential risks at early and mitigation measures can be taken at time– thus effective elimination of the potential causes is possible. It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met.
- **Early Error Detection :** Errors can be detected much earlier as the system is mode side by side.
- **Good and Conductive Project Environment :** Iteration between development team and client provides a very good and conductive environment during project resolve unclear objectives.
- **Better System to Users :** The client and the developers can compare if the software made matches the software specification according to which the software

program is built. It provides a better system to users as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports the tendency.

- **Quicker User Feedback :** The software designer and implementer can obtain feedbacks from the users early in the project leading to better solutions.
- **Communication Interface between user and developers :** Strong Dialog between users and developers are provided through prototyping.
- **Easy Identification of Missing Functionality :** Missing functionalities can be identified easily and confusing or difficult functions can be identified as early as possible.
- **Requirements Validation :** Early involvement of user in the development process ensures requirements validation and quick implementation of incomplete, but functional application.
- **Reduced time and costs :** Prototyping can improve the quality of requirements and specifications provided to developers. Because cost of incorporating changes in the system are exponentially more to implement as they are detected later in development, the early determination user requirement really can result in faster and less expensive software.
- **Improved and increased user involvement :** Prototyping requires user involvements and allows them to see and interact with a prototype. This allow the user to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunication that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product with better quality by satisfying the users to have desire for look, feel and performance.
- **Encourages innovative and Flexible Design :** It encourages innovation and flexible designing.

**Disadvantages of Prototype Model**

Using or perhaps misusing prototyping can also have following disadvantages:

- **Increased Developer Cost :** Prototyping is usually done at the cost of the developer. Sometimes the startup cost of building the development team, focused on making prototype, is high.
- **Slow Process :** It is a slow and lengthy process.
- **Frequent Requirement Change:** Requirements may change frequently that can disturb the rhythm of the development team.
- **Increased Complexity :** Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- **Insufficient analysis :** It may suffer from inadequate problem analysis. The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or poorly engineered final projects. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.
- **User confusion of prototype and finished system :** Too much involvement of user is not always preferred by developers. Users can begin to think that a prototype, intended to be thrown away, is the actual final system that merely needs to be finished or polished. This can lead them to expect the prototype to accurately model the performance of the final system although this is not the intent of the prototyping. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to feature creep.
- **Excessive development time of the prototype :** A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

- **Expectation of Higher Productivity :** A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result.

- **Improper Evaluation :** Approval process and requirement is not strict thus contract may be awarded without rigorous evaluation of Prototype.

- **Difficulty in Documentation :** Identifying and documenting nonfunctional elements are difficult.

**Suitability of Prototype Model**

Prototyping is most beneficial in systems that will have many interactions with the users. In a scenario where there is an absence of detailed information regarding the input to the system, processing needs, output requirements– prototyping model may be employed. It has been found that prototyping is very effective in the analysis and design of on line systems, especially for transaction processing, where the use of screen dialogs is much more in evidence. This type of System Development Method is employed when it is very difficult to obtain exact requirements from the customer. Prototyping is especially good for designing good human computer interfaces. One of the most productive uses of rapid prototyping to date has been as a tool for iterative user requirements engineering and human computer interface design.
Systems with little user interaction, such as batch processing or systems that mostly do calculations benefit little from prototyping.

### 2.5.4 Spiral Model

**History of Spiral Model**

The spiral model was defined by Barry Boehm(23) in his 1988 article "A Spiral Model of Software Development and Enhancement". This model was not the first model to discuss iterative development, but was the first model to explain why the iteration matters. As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with the client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project with an eye toward the

end goal of the project. The schematic diagram of the Spiral model is represented in Figure 2.6.

**Features of Spiral Model**

The Spiral model combines the features of the prototyping model and the waterfall model. This process model proposes incremental development, using the waterfall model for each step with more emphasis on managing risk (23).

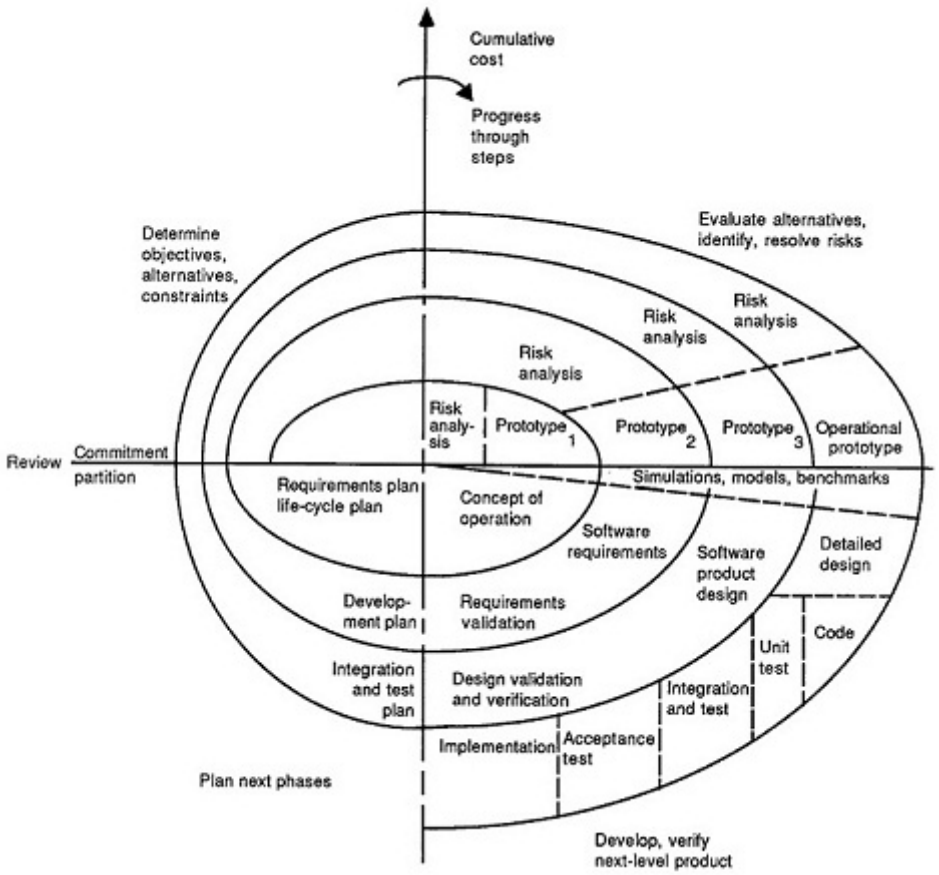


**Figure 2.6:** Spiral Model

The generalized steps in the spiral model are as follows :

- Step 1: The system requirements and other aspects of the systems are defined in much detail by interviewing a number of internal and external users.
- Step 2: A preliminary design is created for the new system.

- Step 3: A first prototype of the new system is constructed from the preliminary design. This is usually a scaled down system which represents an approximation of the characteristics of the final product.
- Step 4: A second prototype is evolved by a four-fold procedure:
    1. Evaluating the first prototype in terms of its strengths, weaknesses, and risks
    2. Defining the requirements of the second prototype
    3. Planning and designing the second prototype
    4. Constructing and testing the second prototype
- Step 5: Based on the customer's option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating cost miscalculation, or any other factor that could result in a less-than-satisfactory final product.
- Step 6: The existing prototype is evaluated in the same manner as was the previous prototype, if necessary another prototype is developed from it according to the fourfold procedure outlined above.
- Step 7: The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product as desired.
- Step 8: The final system is constructed, based on the refined prototype.
- Step 9: The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large scale failures and to minimize downtime.

In spiral model, the aim of customer communication is to establish effective communication between developer and customer. The planning objectives are to define resources, project alternatives, time lines and other project related information. The purpose of the risk analysis phase is to assess both technical and management risks. The engineering task is to build one or more representations of the application. The construction and release task are to construct, test, install and provide user support by documentation and training. The purpose of customer evaluation task is to obtain customer feedback based on the evaluation of the software representation.

**Advantages of Spiral Model**

There are several advantages of spiral model as mentioned below:

- better project risk analysis and avoidance
- strong approval and documentation control
- implementation has priority over functionality
- additional functionalities can be added at a later date
- budget and schedule estimates become more realistic as work progresses as important issues are discovered earlier
- software engineers can get restless with protracted design processes and can start working on the project earlier
- 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

**Disadvantages of Spiral Model**

Inspite of several advantages, the spiral model do have some disadvantages as below:

- Highly customized limiting re-usability
- Applied differently for each application
- Risk of not meeting budget or schedule
- Possibility to end up implemented as the waterfall framework
- Management is more complex
- End of project may not be known early
- Process is complex
- Spiral may go indefinitely
- Large numbers of intermediate stages require excessive documentation
- Doesn't work well for smaller projects

- Can be a costly model to use
- Risk analysis requires highly specific expertise

**Suitability of Spiral Model**

The spiral model is intended for large, expensive, and complicated mission-control projects. It is not suitable for small or low risk projects.

### 2.5.5 V Model

**History of V Model**

The V-Model in shown in Figure 2.7.

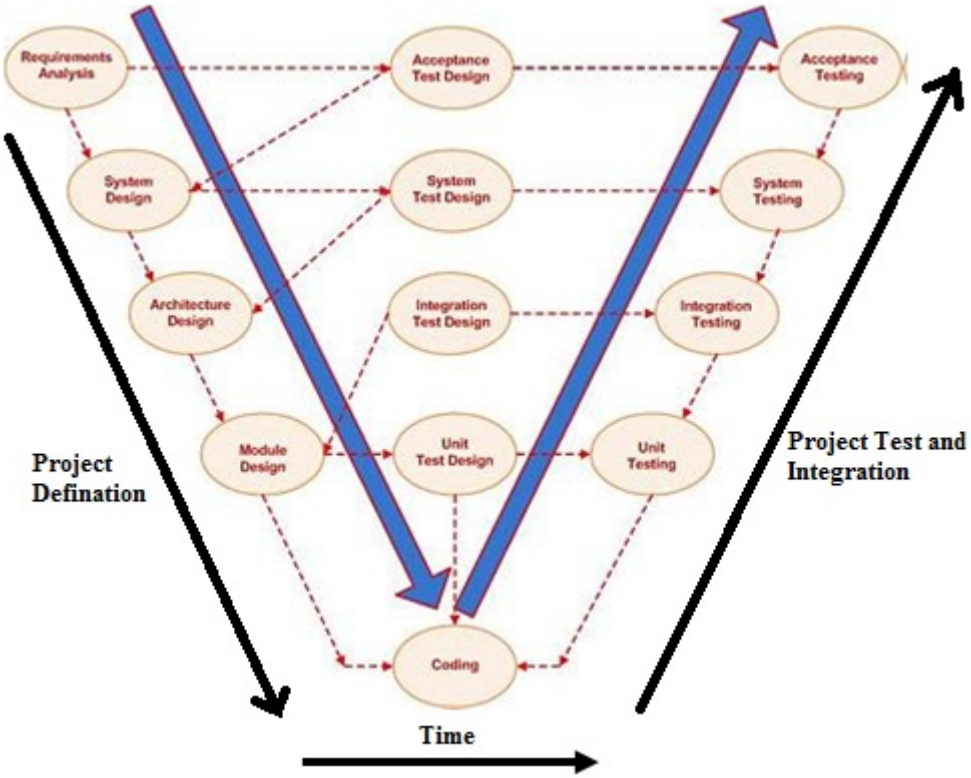


**Figure 2.7:** V (Shape) Model

**Features of V Model**

The V-model is a software development model which can be presumed to be the extension of the waterfall model which emphasizes parallelism of activities of construction and verification. Here, the process steps instead of moving down in a linear way bend upwards after the coding phase resulting in the typical V shape formation. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase,

to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

**Advantages of V Model**

Following are some advantages of V model:

- simple and easy to use
- each phase has specific deliverables
- higher chance of success because of the development of test plans early on during the life cycle
- encourages definition of the requirements before designing the system and designing the software before building the components
- encourages verification and validation of all internal and external deliverables which must be testable, not just the software products
- testing activities are planned before coding which saves time and also helps in developing a very good understanding of the project at the beginning state
- it defines the products that the development process should generate

**Disadvantages of V Model**

The disadvantages of th V process model are:

- it is inflexible
- it is very rigid
- it has no or less ability to respond to change, if present then is difficult and expensive
- if any changes happen in mid way, then the particular documents along with requirement document has to be updated
- it produces inefficient testing methodologies
- software is developed during the implementations phase, so no early prototypes of the software are produced

- doesn't provide a clear path for problems found during testing phases
- doesn't handle concurrent event

**Suitability of V Model**

The V process model works well where requirements are easily understood and best suited for small projects.

### 2.5.6 RAD Model

**History of RAD Model**

Rapid Application Development (RAD) originated from rapid prototyping approaches and was first formalized by James Martin (24) during 1990s, who believed that as compared to traditional lifecycle RAD refers to a development life cycle designed for high quality systems with faster development and lower costs. By the mid 1990s the definition of RAD became used as an umbrella term to encompass a number of methods, techniques and tools by many different vendors applying their own interpretation and approach. This unstructured and extemporized ad hoc evolution of RAD means that the rationale behind its use is not always clear. It is perceived as an IS system methodology, a method for developers to change their development processes or as RAD tools to improve development capabilities (25). RAD projects are sometimes distinguished in terms of intensive and non-intensive forms. A non-intensive approach refers to projects where system development is spread over a number of months involving incremental delivery. Where as, in the intensive RAD project personnel are closeted away to achieve set objectives with a 3–6 week time frame (25). The RAD model is depicted in Figure 2.8.

**RAD Model Phases**

The RAD approach encompasses the following phases:

- Business Modeling:
  The information flow among business functions is defined by answering questions like:
  - what information drives the business process?
  - what information is generated?
  - who generates it?

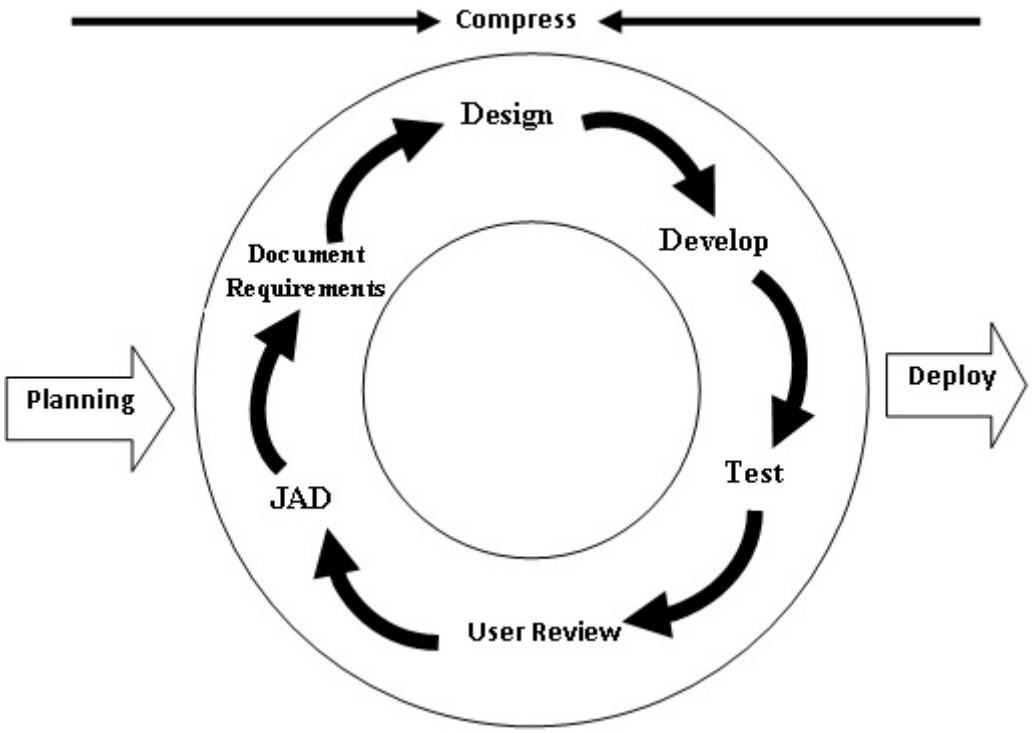


**Figure 2.8:** RAD Model

  - where does the information go?
  - who processes it?

  In these phase, answer to the above questions are explored and gathered.

- Data Modeling:
  The information collected from business modeling is refined into a set of data objects that are needed to support the business. The attributes are identified and the relation between these data objects is defined.

- Process Modeling:
  The data object defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting or retrieving a data object.

- Application Generation:
  Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques. The RAD model assumes the use of the RAD tools like VB, VC++, Delphi etc... rather than creating software using conventional third

generation programming languages. The RAD model works to reuse existing program components or create reusable components. In all cases, automated tools are used to facilitate construction of the software.

- Testing and Turn over:
  Since the RAD process emphasizes reuse, many of the program components have already been tested. This minimizes the testing and development time. But new components must be tested and all interfaces must be fully exercised.

**Features of RAD Model**

Followings are some features of RAD model (24, 26, 27, 28):

- gathering requirements using workshops or focus groups
- deliver the customer's requirements with accuracy using prototyping and short iterations
- early reiterative user testing of designs
- reusing software components
- a rigid schedule that defers design improvements to the next product version
- less formality in reviews and other team communication
- timely and speedy delivery of a working application (in about 20 weeks)
- necessitates the collaboration of small and diverse teams of developers, end users and other stakeholders

**The RAD DEBATE**

As a systems development approach RAD has both critics and supporters whose opinions, in some cases are fundamental to individual philosophies and perceptions of its rationale. Existing literature exposes particular themes of discussion within the RAD arena and a prominent area of debate concerns the scalability of RAD across large and complex environments. Although the lack of provenance is reflected by the limited availability of published material, there is substantial reporting of its application and considerable debate about its appropriateness for different types and sizes of systems development (25, 29, 30). It is further thought that its success is linked to the

project management approach, level of management commitment, degree of end user involvement and the ability of the team to make fast authoritative decisions (31). It is also suggested that RAD projects necessitate cultural and managerial changes because people are required to behave in a different way than in the more structured traditional environments. Consequently without radical shifts in organizational attitudes and structures and peoples'mindsets many projects may fail because the change to new methodologies, methods and techniques did not fit within the culture (32). Hence the potential of a RAD development and delivery approach to meet information systems requirements in uncertain and volatile business settings of complex system development environments is questioned. Critics advocate that the need for high levels of user involvement, stakeholder collaboration, lack of project control and rigor are major issues to its success (24, 26, 27, 33).

**Advantages of RAD Model**

Followings are the advantages of RAD model :

- quick initial reviews are possible
- constant integration isolates problems and encourages customer feedback
- reduces the development time and reusability of components help to speedup development
- all functions are modularized so it is easy to work with

**Disadvantages of RAD Model**

Disadvantages of RAD model are :

- it is based on Object Oriented approach and if it is difficult to modularize the project the RAD may not work well i.e. RAD requires a system that can be modularized
- it requires highly skilled and well trained developers or engineers in the team
- both end customer and developer should be committed to complete the system in a much abbreviated time frame
- if commitment is lacking RAD will fail
- need for high levels of user involvement, stakeholder collaboration

- seeks superior project control

**Suitability of RAD Model**

Where the requirements cannot be locked down in the first few weeks of the project, or where requirements are not available until the later part of the project, even when the requirements push the limits of the technology, RAD is not recommended and cannot be employed. This is also true for research projects where discovery and risk play a big role and are prevalent throughout the project. Thus, RAD process works best in cases where the data is known, the requirements can be defined and kept unchanged during the development and the functional requirements can be met within a short time frame with a small team (3–4 months with 5–6 technical resources). Literature considers it more appropriate for small to medium simple, highly interactive development projects rather than for environments that are also computationally complex such as the case study involved.

### 2.5.7 RUP: Rational Unified Process Model

**History of RUP Model**

The Rational Unified Process (RUP) was brought into the IBM offering by the acquisition of the 20 year old Rational Software Corporation by IBM Software Group in February 2003. The RUP incorporates material in the areas of data engineering, business modeling, project management, and configuration management, the latter as a result of a merger with Pure-Atria in 1997. It includes elements of the Real-Time Object-Oriented Method, developed by the founders of ObjecTime, acquired by Rational in 2000 brings a tighter integration to the Rational Software suite of tools. The Rational Unified Process is the direct successor to the Rational Objectory Process (ROP), version 4, which was the result of the integration of the Rational Approach and the Objectory Process (version 3.8) after the merger of Rational Software Corporation and Objectory AB in 1995. From its Objectory ancestry, the process has inherited its process model and the central concept of use case. From its rational background, it gained the current formulation of iterative development and architecture. This ROP version 4 also incorporated material on requirements management from Requisite, Inc., and a detailed test process inherited from SQA, Inc., companies that also were acquired by Rational Software. Finally, this version of the process was the first to use the newly created Unified Modeling Language (UML 0.8). The Objectory Process was created in Sweden in 1987 by Ivar Jacobson

(34). The Rational Unified Process (RUP) has matured over the years and reflects the collective experience of the many people and companies that today make up the rich heritage of IBM's Rational Software division. Let us take a quick look at the rich ancestry of RUP 2003 (34) as illustrated in the following Figure 2.9.

The Rational Unified Process (RUP) is a software engineering process that provides

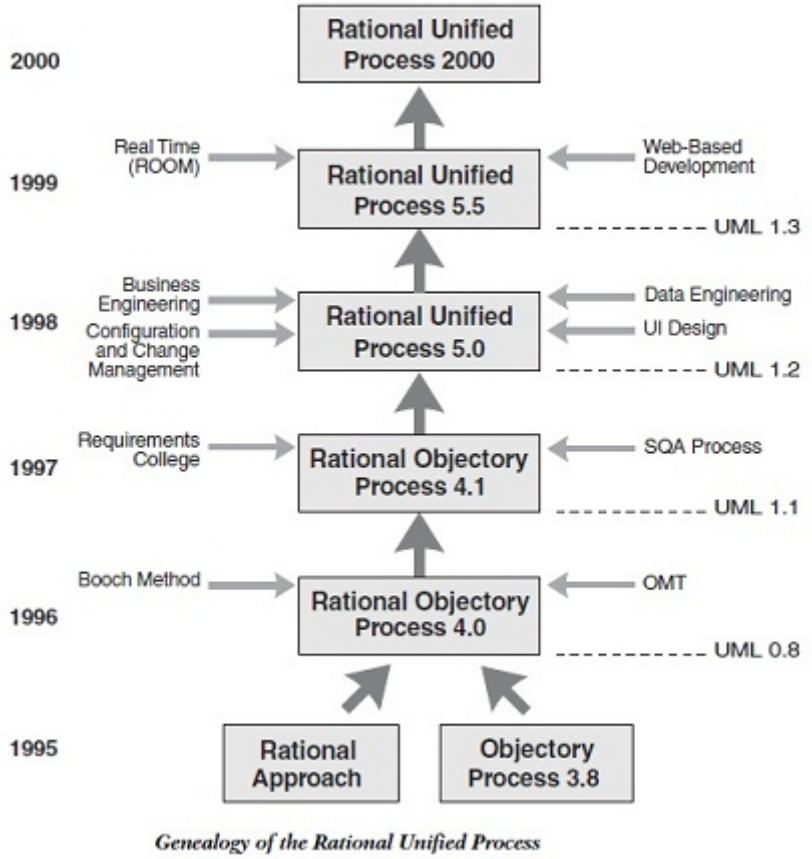


**Figure 2.9:** RUP History

a disciplined approach to assigning tasks and responsibilities within a development organization. The Rational Unified Process is a process product. It is developed and maintained by Rational Software and integrated with its suite of software development tools. The Rational Unified Process is also a process framework that can be adapted and extended to suit the needs of an adopting organization. The following Figure 2.10 (34) shows the structure of the RUP process:

**Features of RUP Model**

The Rational Unified Process captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations as follows :

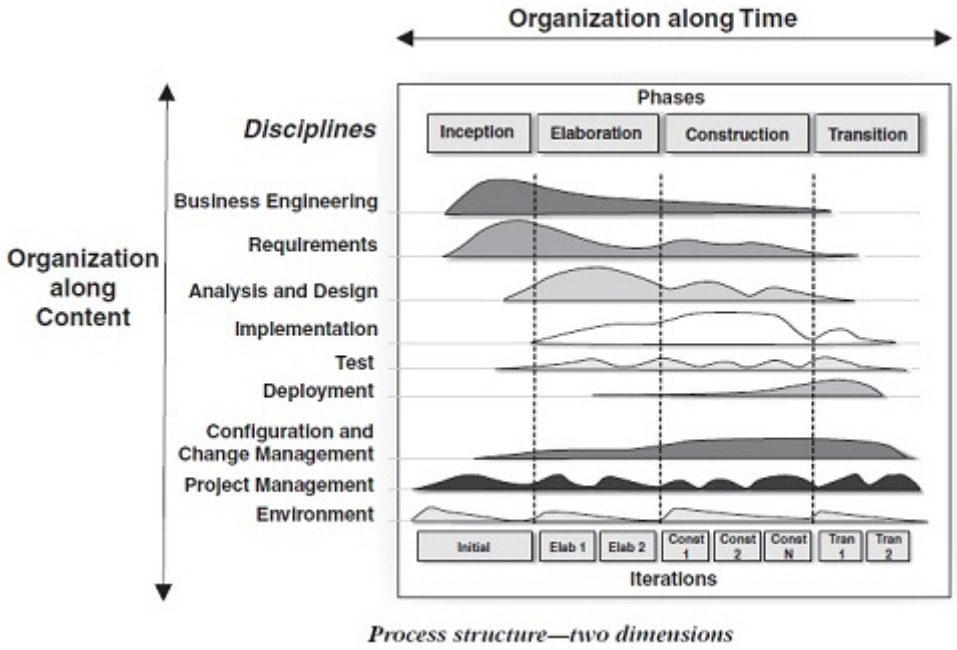


**Figure 2.10:** RUP (Rational Unified Process) Model

- develop software iteratively
- manage requirements
- uses component based architectures
- visually model software
- continuously verify software quality
- control changes to software

**Advantages of RUP Model**

The advantages of the RUP model are as follows :

- a complete methodology to manufacture software
- process with complement document facility
- openly published, distributed and supported
- supports changing requirements to meet its desired software
- supports iteration process so we can integrate the code in development life cycle in lesser time and effort spent in integration

- easy and faster code reuse resulting less development time
- there are online training and tutorials available for this process
- debugging is very easy due to component base architecture
- versatile and wide applicability
- adopts proven best industrial practices

**Disadvantages of RUP Model**

The disadvantages of the RUP model are as follows :

- the team members need to be expert in their field to develop the software under this methodology
- the development process is too complex and disorganized
- on cutting edge projects which utilize new technology, the reuse of components will not be possible and hence the time saving one could have made will be impossible to fulfill
- integration throughout the process of software development, in theory sounds a good thing but on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing
- it is too complex to implement and too difficult to learn
- may lead to undisciplined form of software development

**Suitability of RUP Model**

More than a thousand companies were using the Rational Unified Process at the end of 2000 in various application domains, for both large and small projects. RUP is applicable in various industries like telecommunications, transportation, aerospace, defense, manufacturing, financial services, systems integrators and many more. More than 50% of these users are either using the Rational Unified Process for eBusiness or planning to do so in the near future. The RUP development team at Rational continues to refine the process for the benefit of all.

### 2.5.8 Evolutionary Process Model

Evolutionary development uses small, incremental product releases, frequent delivery to users, dynamic plans and processes. The evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle. The users provide feedback on the product for the planning stage of the next cycle and the development team responds accordingly by changing the product, plans, process etc (35, 36). Figure 2.11 shows the structure of the EVO process(34):

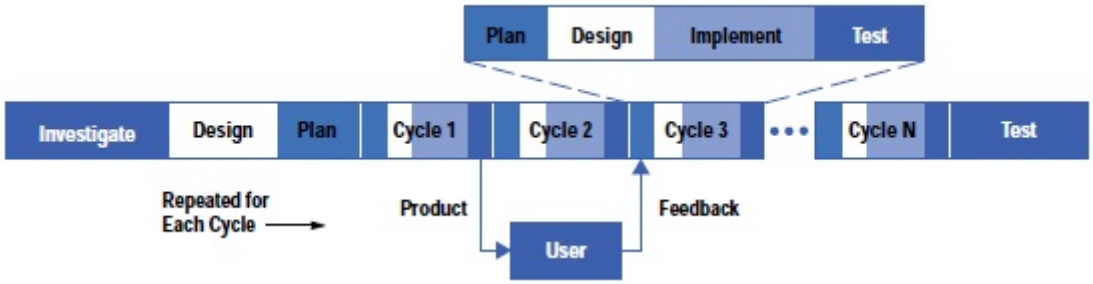


**Figure 2.11:** Evolutionary (EVO) development Process Model

**Advantages of Evolutionary Model**

Successful use of Evolutionary model can benefit not only business results but marketing and internal operations as well. The advantages of the EVO are as follows (35) :

- risk analysis is better-significant reduction in risk for software projects
- supports changing requirements
- initial operating time is less
- can reduce costs by providing a structured, disciplined avenue for experimentation
- the inevitable change in expectations when users begin using the software system is addressed by EVO's early and ongoing involvement of the user in the development process
- allows the marketing department access to early deliveries, facilitating development of documentation and demonstrations

- continuous process improvement becomes a more realistic possibility with one-to-four-week cycles
- during life cycle software is produced early which facilitates customer evaluation and feedback
- the opportunity to show their work to customers and hear customer responses tends to increase the motivation of software developers and consequently encourages a more customer focused orientation
- the cooperation and flexibility required by EVO of each developer results in greater teamwork

**Disadvantages of Evolutionary Model**

The disadvantages of the Evolutionary model are as follows :

- management complexity is more
- end of project may not be known which is a risk
- can be costly to use
- highly skilled resources are required for risk analysis
- project's progress is highly dependent upon the risk analysis phase

**Suitability of Evolutionary Model**

Evolutionary Model is better suited for large and mission-critical projects.