



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

SYLLABUS

II Year- III Semester: B.Tech. (Information Technology)

3IT4-07: Software Engineering

Credit- 3
3L+0T+0P

Max. Marks : 150 (IA:30, ETE:120)
End Term Exam: 3 Hours

SN	CONTENTS	Hours
1	Introduction, software life-cycle models, software requirements specification, formal requirements specification, verification and validation.	8
2	Software Project Management: Objectives, Resources and their estimation, LOC and FP estimation, effort estimation, COCOMO estimation model, risk analysis, software project scheduling.	8
3	Requirement Analysis: Requirement analysis tasks, Analysis principles. Software prototyping and specification data dictionary, Finite State Machine (FSM) models. Structured Analysis: Data and control flow diagrams, control and process specification behavioural modeling	8
4	Software Design: Design fundamentals, Effective modular design: Data architectural and procedural design, design documentation.	8
5	Object Oriented Analysis: Object oriented Analysis Modeling, Data modeling. Object Oriented Design: OOD concepts, Class and object relationships, object modularization, Introduction to Unified ModelingLanguage .	8
TOTAL		40

Office of Dean Academic Affairs
Rajasthan Technical University, Kota

Course Name (Course Code): SOFTWARE ENGINEERING (3IT4-07)

CO	Course Outcomes	Bloom's Level	PO Indicators	PSO Indicators
3IT4-07.1	Understand and compare the various software engineering models and differentiate between verification & validation	2, 5	1.2.1, 1.3.1, 1.4.1, 2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.4.2, 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.6, 3.2.1, 3.2.2, 3.2.3, 4.1.1, 4.1.2, 4.1.4, 4.3.4, 5.1.1, 5.2.1, 5.3.1, 7.1.1, 7.2.1, 7.2.2, 12.1.1, 12.1.2, 12.2.1, 12.2.2, 12.3.1, 12.3.2	1.1.2, 1.1.3, 2.1.1, 2.1.2
3IT4-07.2	Apply various estimation models to determine the cost of software projects and illustrate risks in the software projects	3, 4	1.2.1, 1.3.1, 1.4.1, 2.1.1, 2.1.2, 2.4.1, 2.4.2, 2.4.3, 2.4.4, 3.1.3, 3.1.4, 3.1.5, 3.1.6, 3.2.1, 3.2.2, 3.4.2, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.2.1, 4.2.2, 4.3.1, 5.1.1, 5.1.2, 5.2.1, 5.3.1, 5.3.2, 7.1.1, 7.1.2, 7.2.1, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.2.4, 9.3.1, 11.1.1, 11.1.2, 11.2.1, 11.3.1, 11.3.2, 12.1.1, 12.1.2, 12.2.1, 12.2.2, 12.3.1, 12.3.2	1.1.2, 1.1.3, 2.1.1, 2.1.2, 2.1.3, 3.1.2
3IT4-07.3	Identify the requirements, design the SRS and model various data flow and control flow diagrams	1, 3, 6	1.1.1, 1.1.2, 1.3.1, 1.4.1, 2.1.1, 2.1.2, 2.1.3, 2.2.2, 2.3.1, 2.3.2, 2.4.2, 2.4.1, 2.4.4, 3.1.6, 3.2.1, 3.2.2, 3.4.2, 4.1.1, 4.1.2, 4.1.3, 4.2.1, 4.2.2, 4.3.2, 4.3.3, 5.1.1, 5.1.2, 5.3.1, 7.1.1, 7.2.2, 8.1.1, 8.2.2, 9.1.1, 9.1.2, 9.2.1, 9.2.2, 9.2.3, 9.3.1, 10.1.1, 10.1.2, 10.1.3, 10.2.1, 10.2.2, 10.3.1, 12.1.1, 12.1.2, 12.2.1, 12.2.2, 12.3.1, 12.3.2	1.1.2, 1.1.3, 3.1.2
3IT4-07.4	Learn various approaches to break down software in modules and create documentation	4, 6	1.3.1, 2.1.1, 2.1.2, 2.4.2, 2.4.4, 3.2.1, 3.2.2, 3.2.3, 3.4.2, 4.2.1, 4.2.2, 4.3.1, 4.3.2, 5.1.1, 5.2.1, 5.3.1, 12.1.1, 12.1.2, 12.2.1, 12.2.2, 12.3.1, 12.3.2	
3IT4-07.5	Describe object oriented programming concepts to show class and object relationships	2, 3	1.2.1, 1.3.1, 1.4.1, 2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.3.1, 2.3.2, 2.4.2, 2.4.4, 3.1.6, 3.2.1, 3.2.2, 3.3.1, 3.4.2, 4.1.1, 5.1.1, 5.1.2, 5.2.1, 5.2.2, 5.3.1, 5.3.2	1.1.2, 1.1.3, 3.1.2



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

A

Course File

on

Software Engineering (3IT4-07)

Programme: B.Tech. (Information Technology)

Semester: - III

Session 2021-22

(Ms. Sanju Choudhary)

(Associate Professor)

(Department of Information Technology)



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

UNIT-1

INTRODUCTION TO SOFTWARE ENGINEERING

CONTENTS

Lecture 1: Introduction to Software Engineering

Lecture 2: Software Development Life Cycle

Lecture 3: Classical Waterfall Model, Spiral Model

Lecture 4: Incremental Model, Iterative Waterfall Model, RAD Model

Lecture 5: Prototyping Model, Big-Bang Model

Lecture 6: V-Model, Agile Model

Lecture 7: Software Requirements Specification (SRS), Formal Requirements Specification

Lecture 8: Verification and Validation



INTRODUCTION TO SOFTWARE ENGINEERING

The term *software engineering* is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

So, we can define *software engineering* as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

The product that software professionals build and then support over the long term.

Software encompasses:

- (1) Instructions (computer programs) that when executed provide desired features, function, and performance;
- (2) Data structures that enable the programs to adequately store and manipulate information and
- (3) Documentation that describes the operation and use of the programs.



We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

Without using software engineering principles it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes. Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: **abstraction** and **decomposition**. The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on. Abstraction is a powerful way of reducing the complexity of the problem.

Software products

Generic products: - Stand-alone systems that are marketed and sold to any customer who wishes to buy them. Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

Customized products: - Software that is commissioned by a specific customer to meet their own needs. Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

The economies of ALL developed nations are dependent on software. More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)

Software engineering is concerned with theories, methods and tools for professional software development.

Expenditure on software represents a significant fraction of GNP in all developed countries.

Features of Software:-



• Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered; it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software** - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability**- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost**- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature**- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management**- Better process of software development provides better and quality software product.

CHARACTERISTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:



- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products

SOFTWARE DEVELOPMENT LIFE CYCLE



LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

THE NEED FOR A SOFTWARE LIFE CYCLE MODEL

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure. A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models it becomes difficult for software project managers to monitor the progress of the project.

Different software life cycle models

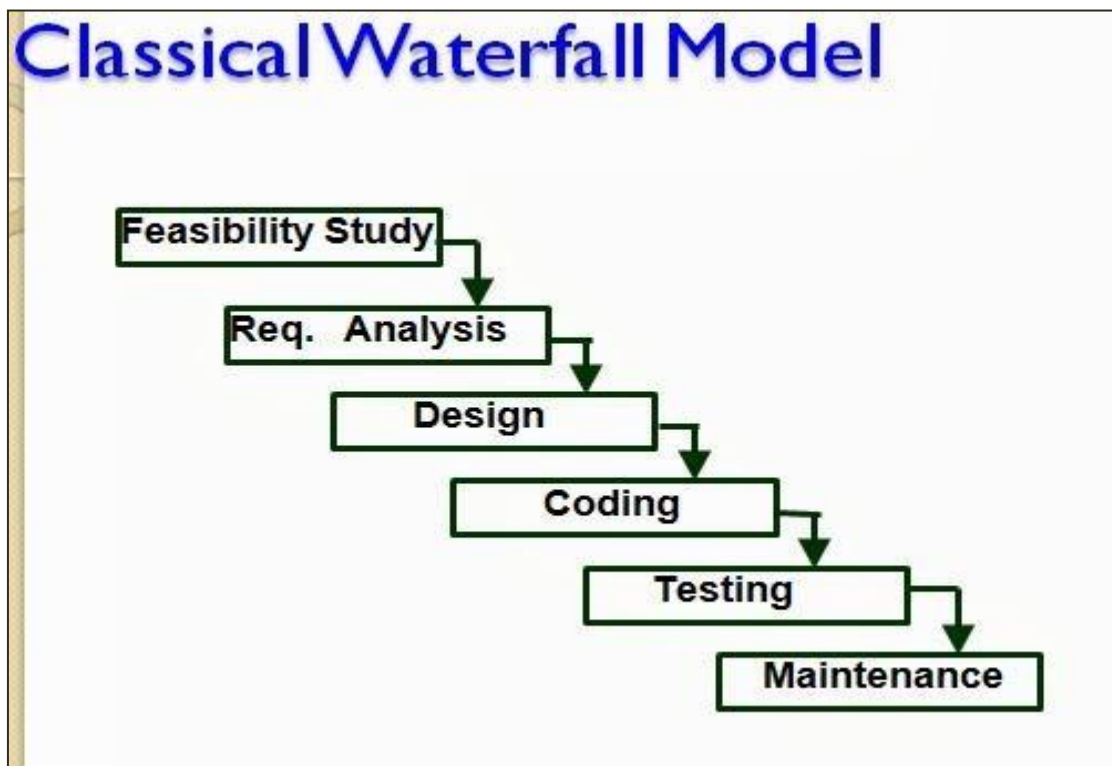
Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:



- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model

1. CLASSICAL WATERFALLMODEL

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 life cycle models it is necessary to learn the classical waterfall model. Classical waterfall model divides the life cycle into the following phases.





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 system and output data to be produced by the system. 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 system.
- 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.

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 system. 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 systematically 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.

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.

- **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.

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.

Integration and system testing: -Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system 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 system is tested and a set of previously planned modules are added to it.



Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

System testing is normally carried out in a planned manner according to the system test plan document. The system 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.

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:

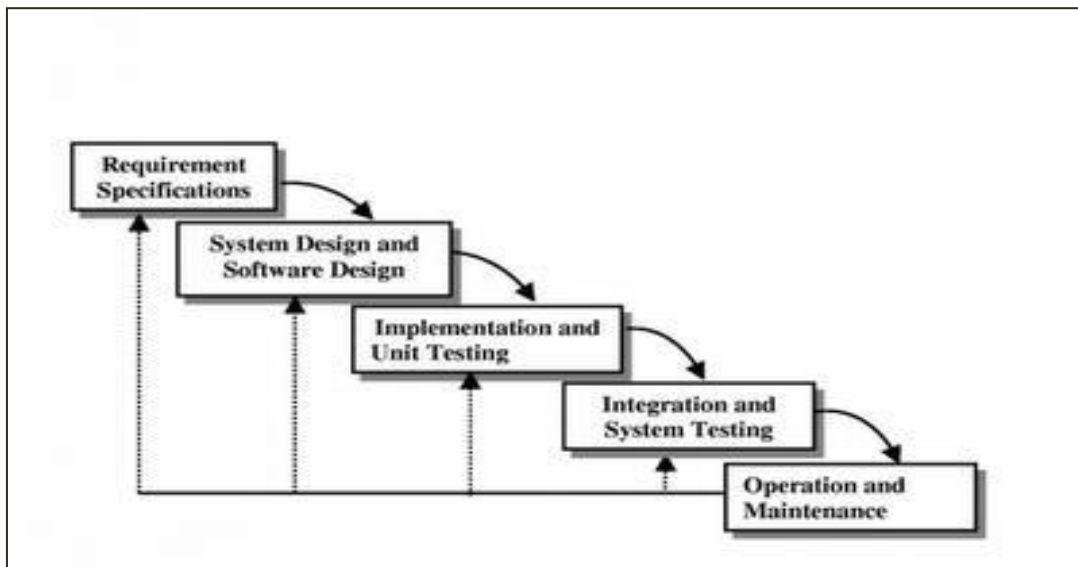
- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting 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 system. This is called adaptive maintenance.

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.

2. ITERATIVE WATERFALL MODEL

To overcome the major shortcomings of the classical waterfall model, we come up with the iterative waterfall model.



Here, we provide feedback paths for error correction as & when detected later in a phase. Though errors are inevitable, but it is desirable to detect them in the same phase in which they occur. If so, this can reduce the effort to correct the bug.

The advantage of this model is that there is a working model of the system 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 system into further small serviceable increments/modules.

3. PROTOTYPING MODEL

A prototype is a toy implementation of the system. A prototype usually exhibits



limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

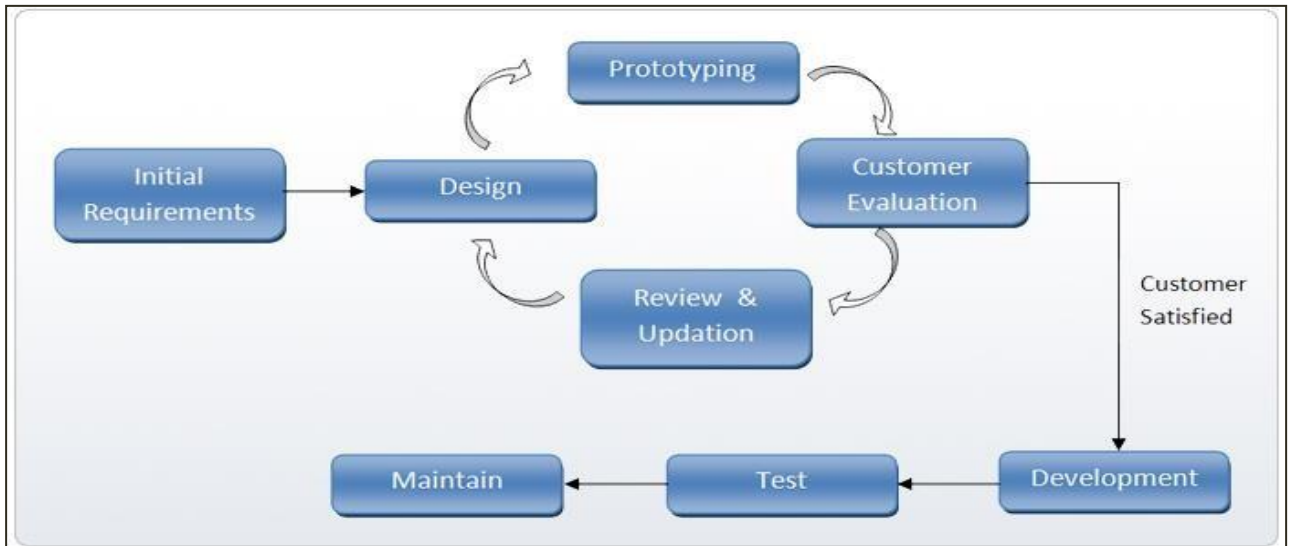
- how the screens might look like
- how the user interface would behave
- how the system would produce outputs

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

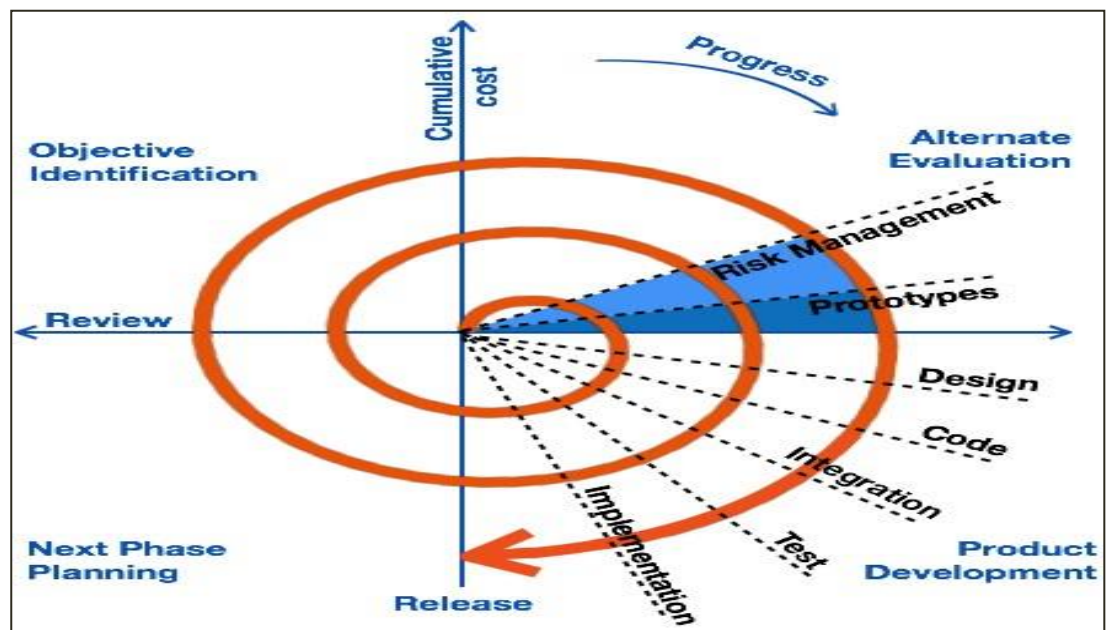
A prototype of the actual product is preferred in situations such as:

- User requirements are not complete
- Technical issues are not clear



4. SPIRALMODEL

The Spiral model of software development is shown in fig. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study, the next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. The following activities are carried out during each phase of a spiral model.





First quadrant (Objective Setting)

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risks associated with these objectives.

Second Quadrant (Risk Assessment and Reduction)

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

- Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Review and Planning)

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

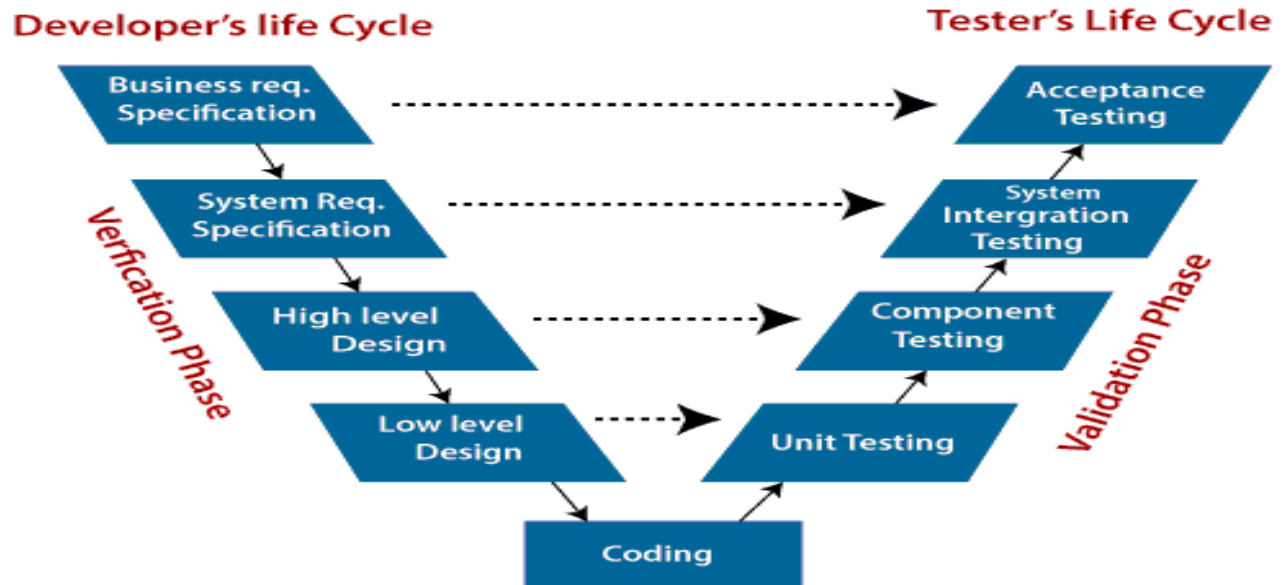
Circumstances to use spiral model

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.

5. V-Model

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

V- Model



Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

Validation: It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

There are the various phases of Verification Phase of V-model:

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a



suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

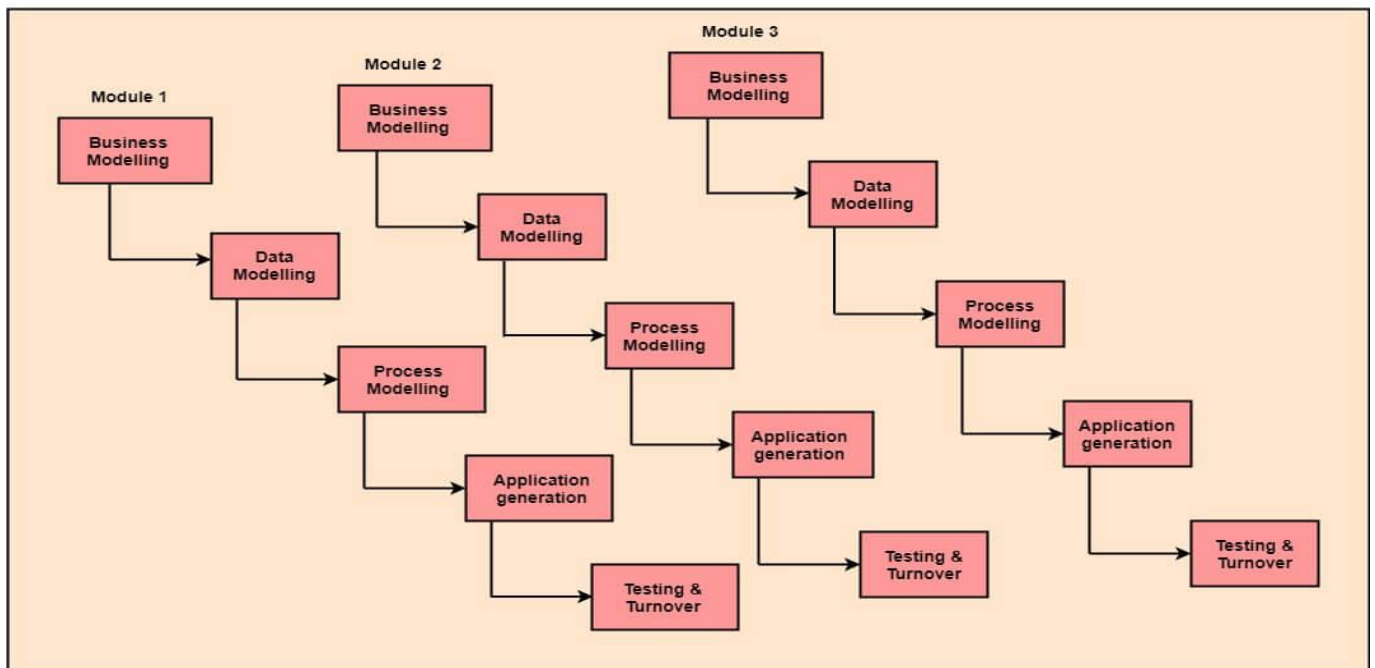
7. RAD (Rapid Application Development) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

Fig: RAD Model



The various phases of RAD are as follows:

1. Business Modeling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modeling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

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

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since



RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

Disadvantage of RAD Model

- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

Big Bang Model

In this model, developers do not follow any specific process. Development begins with the necessary funds and efforts in the form of inputs. And the result may or may not be as per the customer's requirement, because in this model, even the customer requirements are not defined. This model is ideal for small projects like academic projects or practical projects. One or two developers can work together on this model.

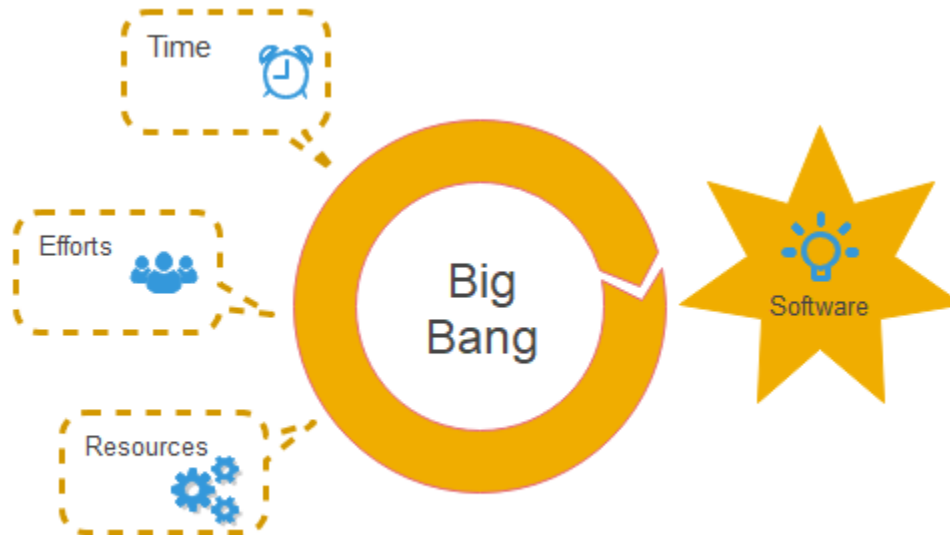


Fig. Big Bang Model

When to use Big Bang Model?

As we discussed above, this model is required when this project is small like an academic project or a practical project. This method is also used when the size of the developer team is small and when requirements are not defined, and the release date is not confirmed or given by the customer.

Advantage(Pros) of Big Bang Model:

1. There is no planning required.
2. Simple Model.
3. Few resources required.
4. Easy to manage.
5. Flexible for developers.

Disadvantage(Cons) of Big Bang Model:

1. There are high risk and uncertainty.
2. Not acceptable for a large project.
3. If requirements are not clear that can cause very expensive.

Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

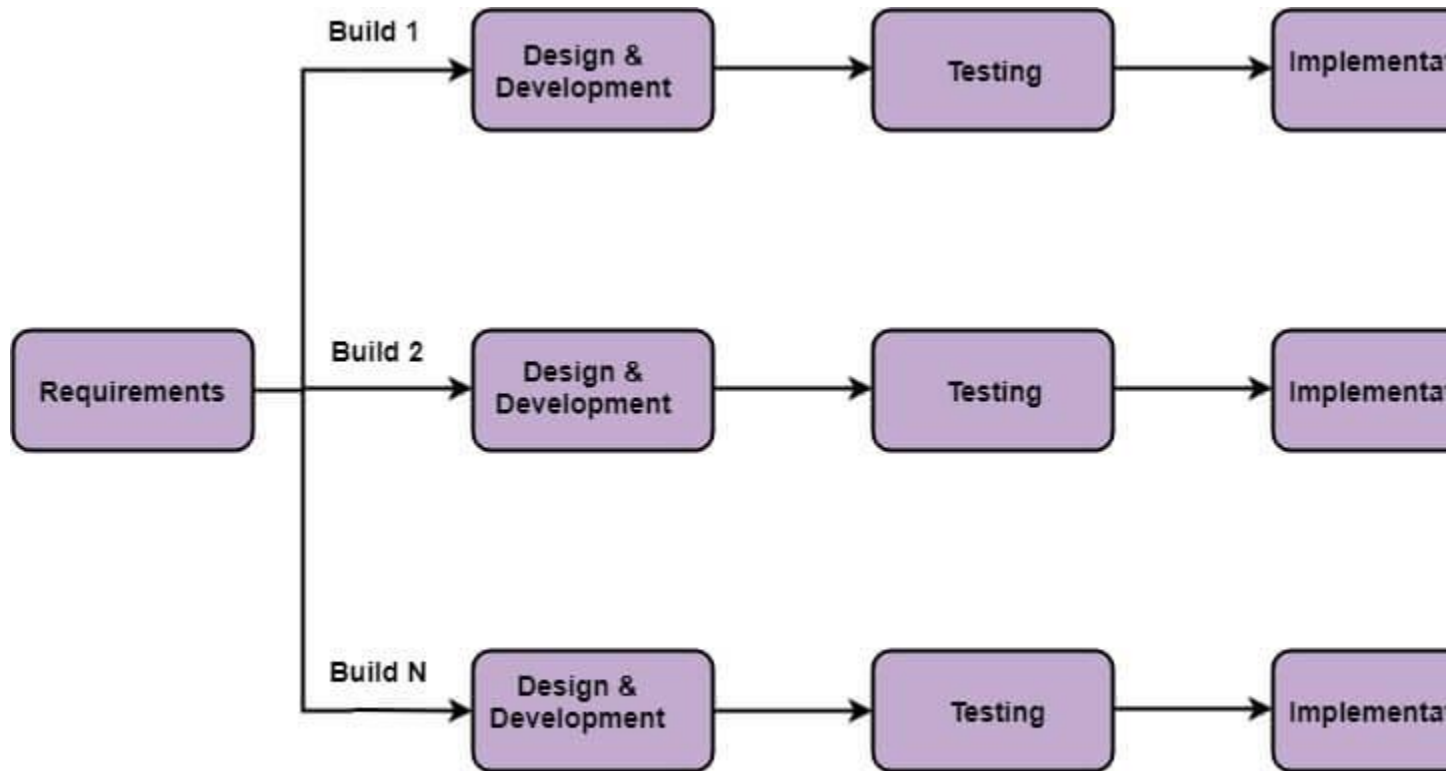


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.



- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

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 system 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



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

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 system. 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.



Software Requirement Specification - [SRS]

What is Software Requirement Specification - [SRS]?

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Types of Requirements:

The below diagram depicts the various types of requirements that are captured during SRS.

How to Write a Software Requirements Specification (SRS Document)

Clear requirements help development teams create the right product. And a software requirements specification (SRS) helps you lay the groundwork for product development.

We'll define what this is, when you'd use one and five steps to writing an SRS Document.

At a glance, this is how to write a requirements document:

- Define the purpose of your product.
- Describe what you're building.
- Detail the requirements.
- Get it approved.

What Is a Software Requirements Specification (SRS) Document?

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfil all stakeholders (business, users) needs.

A typical SRS includes:

- A purpose
- An overall description



Specific requirements

The best SRS documents define how the software will interact when embedded in hardware — or when connected to other software. Good SRS documents also account for real-life users.

Why Use an SRS Document?

A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow.

It's used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page.

Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product's lifecycle — for instance, when to retire a feature.

Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS.

Software Requirements Specification vs. System Requirements Specification

A software requirements specification (SRS) includes in-depth descriptions of the software that will be developed.

A system requirements specification (SyRS) collects information on the requirements for a system.

“Software” and “system” are sometimes used interchangeably as SRS. But, a software requirement specification provides greater detail than a system requirements specification.

How to Write an SRS Document

Writing an SRS document is important. But it isn't always easy to do.

Here are five steps you can follow to write an effective SRS document.

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

1. Create an Outline (Or Use an SRS Template)

Your first step is to create an outline for your software requirements specification. This may be something you create yourself. Or you may use an existing SRS template.

If you're creating this yourself, here's what your outline might look like:

1. Introduction

1.1 Purpose

1.2 Intended Audience

1.3 Intended Use

1.4 Scope

1.5 Definitions and Acronyms

2. Overall Description

2.1 User Needs

2.2 Assumptions and Dependencies

3. System Features and Requirements

3.1 Functional Requirements

3.2 External Interface Requirements

3.3 System Features

3.4 Non-functional Requirements



Once you have your basic outline, you're ready to start filling it out.

2. Start with a Purpose

The introduction to your SRS is very important. It sets the expectation for the product you're building.

So, start by defining the purpose of your product.

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

Intended Audience and Intended Use

Define who in your organization will have access to the SRS — and how they should use it.

This may include developers, testers, and project managers. It could also include stakeholders in other departments, including leadership teams, sales, and marketing.

Product Scope

Describe the software being specified. And include benefits, objectives, and goals. This should relate to overall business goals, especially if teams outside of development will have access to the SRS.

Definitions and Acronyms

It's smart to include a risk definition. Avoiding risk is top-of-mind for many developers — especially those working on safety-critical development teams.

Here's an example. If you're creating a medical device, the risk might be the device fails and causes a fatality.

By defining that risk up front, it's easier to determine the specific requirements you'll need to mitigate it.

3. Give an Overview of what you'll build

Your next step is to give a description of what you're going to build. Is it an update to an existing product? Is it a new product? Is it an add-on to a product you've already created?

These are important to describe upfront, so everyone knows what you're building.

You should also describe why you're building it and who it's for.

User Needs

User needs — or user classes and characteristics — are critical. You'll need to define who is going to use the product and how.

You'll have primary and secondary users who will use the product on a regular basis. You may also need to define the needs of a separate buyer of the product (who may not be a primary/secondary user). And, for example, if you're building a medical device, you'll need to describe the patient's needs.

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

Assumptions and Dependencies

There might be factors that impact your ability to fulfil the requirements outlined in your SRS.



What are those factors?

Are there any assumptions you're making with the SRS that could turn out to be false? You should include those here, as well.

Finally, you should note if your project is dependent on any external factors. This might include software components you're reusing from another project.

4. Detail Your Specific Requirements

The next section is key for your development team. This is where you detail the specific requirements for building your product.

Functional Requirements

Functional requirements are essential to building your product.

If you're developing a medical device, these requirements may include infusion and battery.

And within these functional requirements, you may have a subset of risks and requirements.

External Interface Requirements

External interface requirements are types of functional requirements. They're important for embedded systems. And they outline how your product will interface with other components.

There are several types of interfaces you may have requirements for, including:

- User
- Hardware
- Software
- Communications

System Features

System features are types of functional requirements. These are features that are required in order for a system to function.

Other Nonfunctional Requirements

Non-functional requirements can be just as important as functional ones.

These include:

- Performance
- Safety
- Security
- Quality

The importance of this type of requirement may vary depending on your industry. Safety requirements, for example, will be critical in the medical device industry.

IEEE also provides guidance for writing software requirements specifications.

5. Get Approval for the SRS

Once you've completed the SRS, you'll need to get it approved by key stakeholders. And everyone should be reviewing the latest version of the document.

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in



Structured Analysis and Structured Design (SA/SD)

What is Structured Analysis?

Structured Analysis and Structured Design (SA/SD) is a top-down decomposition technique system design methodology. In software engineering, SA/SD are methods used for analyzing business requirements while developing specifications for converting practices into computer programs, hardware configurations, and related manual procedures.

Structured Analysis is a set of techniques and graphical tools that allow the analyst to develop a new system specification that is easily understandable to the user and a functional, high-quality information system that meets their needs.

Structured Analysis and Structured Design (SA/SD) has a history from way back in the late 1970s modeled by DeMarco, Yourdon, and Constantine after the emergence of the well-known paradigm of modern structured programming. IBM was the first to incorporate Structured Analysis and Structured Design (SA/SD) into its development cycle in the late 1970s and early 1980s.

In contrast, people modified the classical Structured Analysis and Structured Design (SA/SD) due to their inability to represent real-time systems. In 1989, Yourdon came up with another published version of the methodology with a graphical approach known as “Modern Structured Analysis”.

The availability of CASE tools in the 1990s enabled many analysts to develop and modify the graphical Structured Analysis and Structured Design (SA/SD) models. Using this model, analysts attempted to divide a significant, complex problem into smaller, more easily handled ones using a “Divide and Conquer”, “Top-Down approach” (Classical SA), or “Middle-Out” (Modern SA). Analysts used leverage graphics to illustrate their ideas whenever possible to depict a functional view of the problem and maintain relevant written records.

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes –

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development.

They are –

- Data Flow Diagrams



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode

Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user want a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

Symbol

Name

Symbol Meaning

Square Source or Destination of Data

Arrow Data flow

Circle Process transforming data flow

Open

Rectangle

Data Store

The following table shows the symbols used in designing a DFD and their significance



Types of DFD

Department of Information Technology, SKIT, Jaipur – 302017, Rajasthan (INDIA)

URL: www.skit.ac.in

DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

Physical DFD Logical DFD

It is implementation dependent. It shows which functions are performed.

It is implementation independent. It focuses only on the flow of data between processes.

It provides low level details of hardware, software, files, and people.

It explains events of systems and data required by each event.

It depicts how the current system operates and how a system will be implemented.

It shows how business operates; not how the system can be implemented.

Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Text Books

- Jessica Keyes. *Software Engineering Handbook*. Auerbach Publications (CRC Press), 2003.
Contains complete examples of various SE documents.
- Roger S. Pressman. *Software Engineering: A Practitioner's Approach (Sixth Edition, International Edition)*. McGraw-Hill, 2005.
- Ian Sommerville. *Software Engineering (Seventh Edition)*. Addison-Wesley, 2004.
- Hans van Vliet. *Software Engineering: Principles and Practice (Second Edition)*. Wiley, 1999.

Reference Books

- Timothy C. Lethbridge & Robert Laganière. *Object-Oriented Software Engineering: Practical Software Development using UML and Java (Second Edition)*. McGraw-Hill, 2005.
- M.R.V. Chaudron, J.F. Groote, K.M. van Hee, C. Hemerik, L.J.A.M. Somers and T. Verhoeff. "*Software Engineering Reference Framework*". Technical Report CS-Report 04-039, *Computer Science Reports*, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.
- Ian K. Bray. *An Introduction to Requirements Engineering*. Pearson Addison Wesley; 1st edition (August 26, 2002).
- Alan M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice Hall PTR; 2nd Revised edition (March 1993).

Prerequisites of Software Engineering Course

Before start the software engineering course student should have knowledge about following topics:-

1. System Model



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagaria, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Teaching-Learning Methodology

Process followed to improve quality of Teaching Learning.

- Adherence to Academic Calendar
- Use of various instructional methods and pedagogical initiatives

1. **Lecture:** Primarily a method of delivering course content in the class (online/offline) and preferably used for every course.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**


Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

SE Quiz

 sanju@skit.ac.in (not shared) [Switch account](#)



* Required

Name of Student *

Your answer

Roll No. *

Your answer

Class & Section *

Your answer

Following diagrams as a type of Class diagram, component diagram, object diagram, and deployment diagram? * 1 point

- Non-behavioral
- Structural
- Non structural
- Behavioral



UML diagrams are used to: *

1 point

- Program only in Java
- API for all classes
- Executable logic to reuse across classes
- Specify required services for types of objects

Which of the following UML diagrams has a static view? *

1 point

- Activity
- Collaboration
- State chart
- Use case

How many diagrams are there in UML. *

1 point

- 6
- 9
- 8
- 12



Which of the following is not a UML diagram? *

1 point

- Object Diagram
- Interface diagram
- Use case model
- Class diagram

Activity diagram, use case diagram, collaboration diagram, and sequence diagrams are? *

1 point

- non-structural
- structural
- behavioral
- non-behavioral

Which diagram is used to show interactions between messages are classified as? *

1 point

- Collaboration
- Object lifeline
- State chart
- Activity



DFD describe? *

1 point

- Flow of data
- All of the above
- Entity
- Processes

The horizontal line of a sequence diagram shows *

1 point

- line
- Abstract
- Messages
- time

The recurring aspects of designs are called design. *

1 point

- Documents
- Patterns
- Methods
- Structures



What is the first step in the software development lifecycle? *

1 point

- Preliminary Investigation and Analysis
- System Design
- Coding
- System Testing

What does the study of an existing system refer to? *

1 point

- System Analysis
- Feasibility Study
- Details of DFD
- System Planning

Details of DFD Feasibility Study System Analysis System Planning *

1 point

- Structure Analysis
- List
- Plan
- Algorithm
- Other:



Which of these are the various techniques to generate design alternatives? *

1 point

- Determine Component based quality attribute
- Determine Functional Component
- Modify an existing architecture
- All of the mentioned

Software is defined as _____ *

1 point

- Set of programs
- Set of programs, documentation & configuration of data
- Documentation and configuration of data
- None of the mentioned

Functional components for a working models can be stated as which of the following? *

1 point

- All of the mentioned
- Configuring Process Start up
- Providing User interface
- Allowing user to monitor and repair the system



What are the features of Software Code? *

1 point

- All of the above
- Simplicity
- Accessibility
- Modularity

SDLC Model selection is based on _____. *

1 point

- Requirements
- Development team & users
- Project type & associated risk
- All of the above

When the user participation isn't involved, which of the following models will not result in the desired output? *

1 point

- Prototyping & Waterfall
- Prototyping & RAD
- RAD & Spiral
- Prototyping & Spiral



Which of the following model will be preferred by a company that is planning to deploy an advanced version of the existing software in the market? *

1 point

- RAD
- Both (b) and (c)
- Spiral
- Iterative Enhancement

Which of the following falls under the category of software products? *

1 point

- Embedded, CAM
- Customized, Generic
- CAD, Embedded
- Firmware, CAD

Which of the following refers to internal software equality? *

1 point

- Reusability
- Scalability
- Reliability
- Usability



RSA is abbreviated as _____, invented by a division of _____. * 1 point

- Rational Software Architect, IBM
- Rational Software Architect, Infosys
- Rational Software Architecture , IBM
- Rational software analysis, Infosys

The Nonfunctional components are _____ * 1 point

- Adaptability
- Reliability
- All of the mentioned
- Re usability

Which one of the following activities is not recommended for software processes in software engineering? * 1 point

- Software Evolution
- Software designing
- Software Testing & Validation
- Software Verification



Arrange the following activities to form a general software engineering process model. I. Manufacture II. Maintain III. Test IV. Install V. Design VI. Specification *

1 point

- 6, 5, 1, 3, 4, 2
- 1, 6, 5, 2, 3, 4
- 1, 2, 4, 3, 6, 5
- 6, 1, 4, 2, 3, 5

The _____ model helps in representing the system's dynamic behavior. *

1 point

- Data Model
- Object Model
- Context Model
- Behavioral Model

Which of the following word correctly summarized the importance of software design? *

1 point

- Complexity
- Efficiency
- Accuracy
- Quality



What does a data store symbol in the Data Flow Diagram signify? *

1 point

- Data Structure
- Physical File
- All of the above
- Logical File

_____ is not a direct measure of SE process. *

1 point

- Cost
- Effort
- All of the above
- Efficiency

Which parameters are essentially used while computing the software development cost? *

1 point

- Hardware and Software Costs
- All of the above
- Training Costs
- Effort Costs



Which of the following threatens the quality and timeliness of the produced software? *

1 point

- Business risks
- Technical risks
- Potential risks
- Known risks

What is a collection of model elements called? *

1 point

- UML packages
- Package members
- Dependency
- Box

What does a component diagram consists of? *

1 point

- Packages and dependency
- Internal structure, Components & their Relationship to the environment
- Components, their Relationship to the environment
- Internal structure



Component diagrams commonly contain components, interfaces and _____* 1 point

- relationships
- nodes
- objects
- classifiers

In deployment diagram, a node is represented as a _____* 1 point

- prism
- cuboids
- rectangular
- cube

_____ relationship is used among nodes in deployment diagram.* 1 point

- Generalization
- Aggregation
- Dependency
- Association



Source code files and data files are contained by the _____ components *

1 point

- system
- execution
- deployment
- work product

._____ represented by In UML diagrams, relationship between component parts and object. *

1 point

- ordination
- aggregation
- segregation
- increment

Multiplicity for an association _____.*

1 point

- association is the number of instances with a single instance
- association is the number of instances with a number instance
- All of the mentioned
- None of the mentioned



_____ shows how a system will be physically deployed in the hardware environment * 1 point

- Component diagram
- Deployment diagram
- Class diagram
- Use case diagram

On what basis is plan-driven development different from that of the software development process? * 1 point

- Based on the iterations that occurred within the activities.
- Based on the output, which is derived after negotiating in the software development process.
- Based on the interleaved specification, design, testing, and implementation activities.
- All of the above

The agile software development model is built based on _____. * 1 point

- Incremental Development
- Both Incremental and Iterative Development
- Iterative Development
- Linear Development



An erroneous system state that results in an unexpected system behavior 1 point
is acknowledged as? *

- System failure
- Human error or mistake
- System error
- System fault

What does a directed arc or line signify in DFD? *

1 point

- Data Flow
- Data Process
- Data Stores
- None of the above

What is the main task of project indicators? *

1 point

- To evaluate the ongoing project's status and track possible risks.
- To evaluate the ongoing project's status.
- To track potential risks.
- None of the above



Which diagram in UML shows a complete or partial view of the structure of a modeled system at a specific time? * 1 point

- Sequence Diagram
- Collaboration Diagram
- Class Diagram
- Object Diagram

What is the main intent of project metrics in project libre tool? * 1 point

- For strategic purposes
- To minimize the development schedule.
- To evaluate the ongoing project's quality on a daily basis
- To minimize the development schedule and evaluate the ongoing project's quality on a daily basis

Which of the following is an incorrect activity for the configuration management of a software system? * 1 point

- Change management
- System management
- Internship management
- Version management



The project planner examines the statement of scope and extracts all-important software functions, which is known as *

1 point

- Planning process
- Decomposition
- Association
- All of the mentioned

Which of the following is not included in the total effort cost? *

1 point

- Costs of lunch time food
- Costs of support staff
- Costs of networking and communications
- Costs of air conditioning and lighting in the office space

Which of the following is used to predict the effort as a function of LOC or FP? *

1 point

- COCOMO
- FP-based estimation
- Both COCOMO and FP-based estimation
- Process-based estimation



Once the requirements are stabilized, the basic architecture of the software can be established. Which of the following version of the COCOMO model conforms to the given statement? *

1 point

- Application composition model
- Post-architecture-stage model
- Early design stage model
- All of the above

Which of the following refers to the systematic attempt, which is implemented to ascertain the threats to any project plan? *

1 point

- Performance risk
- Risk identification
- Risk projection
- Support risk

Interaction Diagram is a combined term for *

1 point

- Sequence Diagram + Collaboration Diagram
- Activity Diagram + State Chart Diagram
- Deployment Diagram + Collaboration Diagram
- None of the mentioned

[Submit](#)[Clear form](#)

Never submit passwords through Google Forms.



This form was created inside of Swami Keshvanand Institute of Technology. [Report Abuse](#)



Google Forms





List of Software Engineering (3IT4-07) Text Books and Reference Books

Text Books

- **Jessica Keyes.** *Software Engineering Handbook.* Auerbach Publications (CRC Press), 2003.
Contains complete examples of various SE documents.
- **Roger S. Pressman.** *Software Engineering: A Practioner's Approach (Sixth Edition, International Edition).* McGraw-Hill, 2005.
- **Ian Sommerville.** *Software Engineering (Seventh Edition).* Addison-Wesley, 2004.
- **Hans van Vliet.** *Software Engineering: Principles and Practice (Second Edition).* Wiley, 1999.

Reference Books

- Timothy C. Lethbridge & Robert Laganière.
Object-Oriented Software Engineering: Practical Software Development using UML and Java (Second Edition). McGraw-Hill, 2005.
- M.R.V. Chaudron, J.F. Groote, K.M. van Hee, C. Hemerik, L.J.A.M. Somers and T. Verhoeff. "Software Engineering Reference Framework". Technical Report CS-Report 04-039, *Computer Science Reports*, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.
- Ian K. Bray. *An Introduction to Requirements Engineering.* Pearson Addison Wesley; 1st edition (August 26, 2002).
- **Alan M. Davis.** *Software Requirements: Objects, Functions, and States.* Prentice Hall PTR; 2nd Revised edition (March 1993).

Assignment-2

Part-A(ANSWER UPTO 25 WORDS)

1. Define Verification and validation.
2. Write the Objective Of Software Project Planning.
3. Two Differences Between LOC and FP estimation.
4. List the Requirement Analysis Task.
5. Why accuracy is important in data dictionary.
6. How are coupling and software portability related to each other.
7. What is the difference between Design walkthrough and Design Inspection.
8. Can we have Inheritance without polymorphism.
9. List Object Oriented Design approaches.
10. Define UML.

Part-B

1. Explain typically three types of risk that software can suffer from.
2. Explain Components Of SRS.
3. Write the IEEE recommended structure.
4. Explain Effective Modular Design.
5. Describe Object Modularization.

Part-C

1. Explain RAD Model in detail.
2. Explain: Stepwise Refinement, Modularity, and Information Hiding.
3. Describe Object Oriented Analysis Modeling.
4. Draw a Data Flow Diagram for Traffic Control System,0-level,1-level and 2-level.